An Arabic Sentence Generator

by

Safran Ali Al-Safran

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

July, 1992

Order Number 1354053

An Arabic sentence generator

Al-Safran, Safran Ali, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1992

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

AN ARABIC SENTENCE GENERATOR

BY

SAFRAN ALI AL-SAFRAN

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

JULY 1992

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by Safran Ali Yousif Al-Safran under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in COMPUTER SCIENCE.

THESIS COMMITTEE

Thesis Advisor

Member

Member

Member

Department Chairman

Dean, College of Graduate Studies

Date : 4-7-92

To
MY Mother, Father, Brothers, and Sisters

ACKNOWLEDGMENTS

Akcknowledgment is due to the King Fahd University of Petroleum and Minerals for providing the opportunity to carry out this research work.

I would like to express my appreciation to my thesis advisor ***DR. Mohammed Gazaly Khayat*** for his patient guidance, encouragement, and tremendous help.

I would also like to express my thanks to the other members of my theis Committe ***Dr. Mostafa Aref, Dr. Muhammed Al-Mulhem,*** and ***Dr. Kanaan Faisal,*** for their helpful comments.

THESIS ABSTRACT

NAME OF THE STUDENT: **SAFRAN ALI AL-SAFRAN**

THESIS TITLE : **AN ARABIC SENTENCE GENERATOR**

MAJOR FIELD : **COMPUTER SCIENCE**

DEGREE DATE : **JULY 1992**

This research work is part of an on-going research project on natural Arabic understanding at KFUPM. The project is aimed at building "A Natural Arabic Understanding System" (NAUS). Our aim is to develop a sentence generation module for the system.

This research work presents the design, implementation and integration of an Arabic Sentence Generator (ASG) within NAUS. ASG consists of four main components: preprocessor module, morphological module, end-case module, and syntactic module. The design and implementation of this system are modular.

In the implementation, the morphological, end-case, and syntactic modules have been integrated with the correspond analysis modules of NAUS. This, in turn, reduced development effort and minimizes program size. ASG has been implemented in Prolog.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

July, 1992

خلاصة الرسالة

اسم الطالب : صفران علي يوسف الصفران

عنوان الدراسة : نظام آلى لأنتاج الجمل العربية

التخصص : علوم الحاسب الآلى

تاريخ الدرجة : يوليو ١٩٩٢ م

يمثل هذا البحث جزء من "النظام الآلى لفهم اللغة العربية" الذى يتم تطويره في جامعة الملك فهد للبترول والمعادن . ويختص البحث بتطوير نظام انتاج الجمل العربية.

يقدم هذا البحث تصميم وبناء نظام آلي لأنتاج الجمل العربية بناء على قواعد النحو والصرف والاعراب. ويعتمد تصميم النظام وبناؤه على تقسيمه الى أربع وحدات مترابطة هي : وحدة المعالجة الأولية ، ووحدة التركيب الصرفي ، ووحدة التركيب النحوي ، ووحدة الاعراب.

وقد تم دمج وحدات التركيب مع وحدات التحليل المقابلة في نظام فهم اللغة العربية لتسهيل عملية بناء المركب الصرفي والنحوي والاعرابي . وقد تم بناء نظام انتاج الجمل العربية باستخدام لغة برولغ.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران - المملكة العربية السعودية

TABLE OF CONTENTS

THESIS ABSTRACT (English) v
THESIS ABSTRACT (Arabic) vi
TABLE OF CONTENTS vii

CHAPTER I : INTRODUCTION 1

1.1 Introduction 1
1.2 Problem Definition 4
1.3 Research in Natural Language Generation 5
1.4 Research in Arabic Sentence Generation 9

CHAPTER II : ARABIC SENTENCE GENERATOR (ASG) 12

2.1 Introduction 12
2.2 Design 12
2.2.1 Preprocessor Module 13
2.2.2 Morphological module 17
2.2.3 Syntactic Module 22
2.2.4 End-Case Module 26
2.3 The Interaction Between the Modules 48
2.4 Embedding the Arabic Sentence Generator into NAUS 50

CHAPTER III : IMPLEMENTATION OF ASG 56

3.1 Preprocessor 56
3.2 Morphology 56
3.2.1 The Verb Submodule 56
3.2.2 The Noun Submodule 59
3.2.3 The Article Submodule 61

3.3 End-Case 63
3.3.1 The Noun, Verb, and Article submodules 63
3.4 Syntax 65
3.4.1 The Syntactic Analyzer. 65
3.5 Examples. 69

CHAPTER IV : CONCLUSION AND FUTURE WORK 82

APPENDIX(A): Program Sample Segments 85

REFERENCES 112

LIST OF FIGURES

Figure		*Page*
1.1	NAUS Components	3
2.1	List of Special Article Predicates	14
2.2	Morphological Module Design	18
2.3	Sentence Classifications	24
2.4	Design Structure of the End-Case Module	27
2.5	Verb Types	39
2.6	Verb Irab Categories	40
2.7	Present Verb Irab Categories	41
2.8	Article Types	46

LIST OF TABLES

Table		*Page*
2.1	Verb Derivations	21
2.2	Noun Derivations	21
2.3	Sentence End-Case Classification	29
2.4	Variable & Invariable Nouns	31
2.5	Variable Nouns & their End-Case Marks	32
2.6	Noun Irab Cases & the Corresponding End-Case Positions.	35
2.7	Variable Nouns & their End-Case Marks	36
2.8	Verb Irab Cases & the Corresponding End-Case Positions.	44

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION

Natural language communication with computers has long been a major goal of AI both for the information it can give about intelligence in general and for its practical utility.

This research work is part of an on-going research project on natural Arabic understanding at KFUPM [1,2,4,15]. The project is aimed at building "A Natural Arabic Understanding System" (NAUS). NAUS has the following components: morphology analyzer, syntax analyzer, end-case analyzer, semantics, and sentence generation [1,2,4,15] (Figure 1.1). The morphological, syntax, end-case analysis modules have been fully developed [1,2,4]. Our aim is to develop a sentence generation module for the system.

Sentence generation can be thought of as the process of deliberately constructing a natural-language text in order to meet specified communicative goals. The issues of sentence generation have been categorized as follows: text planning, realization grammar, and lexical choice [14,19]. Most of the work in generation has focused on the syntactic issues of realization: how to represent rules of grammar, what types of grammars are computationally useful, and how to traverse input representations to produce language [14]. In contrast, the other issues have received less attention.

Text planning is thought of as the process of determining which elements to say, structuring the input elements, building noun phrases from a set of attributes, determining the appropriate level of detail to use, controlling the slant and style of the text, and so on [14]. Lexical choice is viewed in different ways. For example, McDonald [14] takes the view that the selection of key lexical items is the first step in the generation process. However, Matthiessen [17] views the lexical choice as part of the unified problem of lexicogrammatical choices.

Chapter 1 of this thesis presents the problem definition, overview of current research in natural language generation, and current research in Arabic sentence generation. Chapter 2 covers the design of the Arabic sentence generator (ASG), the interaction between the modules, and embedding ASG into NAUS. Chapter 3 covers the implementation of ASG, and some examples. Chapter 4 contains the conclusion.

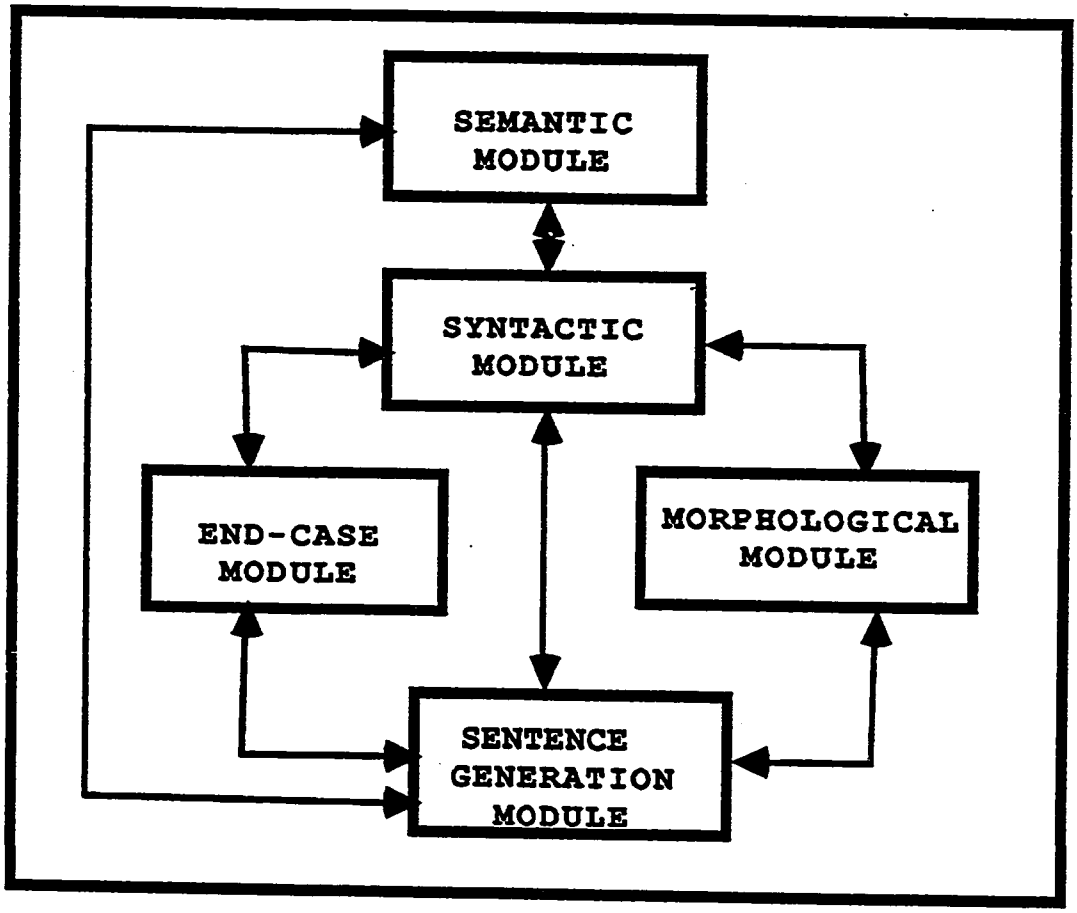

Figure 1.1 : NAUS COMPONENTS

1.2 PROBLEM DEFINITION

The problem is to design and implement an Arabic sentence generator (ASG) of Arabic sentences. The generator is to be integrated into a natural arabic understanding system (NAUS) under development.

The input to ASG is a sentence meaning, and the output is a correct Arabic sentence. More specifically, the input to the ASG includes the following:

1. the representation of the sentence meaning which includes the syntactic and morphological constituents of the sentence.
2. the following information for each construct in the sentence:
   - type (noun, verb, article),
   - type of noun (plural, dual, singular, masculine, feminine),
   - tense of verb (past, present, future),
   - type of article (interrogative,.....etc), and
   - root, and derivation.

The output of the ASG should satisfy the following requirements :

- correct syntactic and morphological constructions,
- correct agreement in number and gender between verbs and nouns, and between subjects and qualifiers, and
- correct end-cases of words.

Before embarking to present a solution, we first survey relevant research efforts in the area. We subdivide the efforts into two classes: current research in natural language generation, and current research in Arabic sentence generation.

1.3 RESEARCH IN NATURAL LANGUAGE GENERATION

Natural language generation is the process of constructing phrases, sentences, or paragraphs. In a sense, it is the opposite of natural language understanding. Although this problem has been investigated for some years, few principles have emerged, and the approaches have varied widely.

Generation can be thought of as generating random sentences to test a grammar, or converting information from an internal representation into a natural language [6].

With respect to random generation, Victor Yngve's system [6] was one of the first research efforts to generate English text randomly. It used a generative context-free grammar and a random-number generator to produce "grammatical" sentences. The system randomly selected one production from among those that were applicable at each point in the generation process, starting from those productions that "produced" *<SENTENCE>* and finally randomly selecting words to fill in the *<NOUN>*, *<VERB>*, and other such positions.

Another system is the Friedman's system [6], which was designed to test the effectiveness of transformational grammars. It operated by generating phrase markers (derivation trees) and by performing transformations on them until a surface structure was generated. The generation is random, but the user could specify an input phrase marker and semantic restriction between various terminals in order to test specific rules for grammatical validity.

The general goal of natural language generation in AI is to take some internal representation of the meaning of a sentence and convert it to a surface-structure form, that is, into an appropriate string of words. There has been considerable variety among such systems, reflecting differences both in the type of internal representation used and in the overall purpose for which the text is generated.

Neil Goldman's program [6] generates sentences from a database of conceptual dependency networks (CD), for which the knowledge representation scheme of this conceptual dependency is based on language-independent semantic primitives. The selection of the actual word for the output is performed by a text-generation subsystem (called BABEL), which uses a discrimination net (a kind of binary decision tree) to select an appropriate verb sense to represent the event specified by the CD. Once a verb sense has been selected, an associated framework that includes information concerning the form of the net and where in the conceptualization the necessary information is located, is used to generate a case-oriented syntax net. After the framework has been filled out, other language-specific functions operate on the syntax net to complete it syntactically with respect to such information as tense, form, mood, and voice. Finally, an ATN is used to generate the surface structure [6].

Another generator, SEMSYN [21], has been applied to a variety of generation tasks within text generation. It is organized into two major modules: the generator kernel (realization component), and the front end generator (morpho/syntactic component). The input to the generator kernel is a semantic representation, i.e. a 'message' derived from Japanese, which will be realized to

express its content in natural language, and the output is a functional grammatical structure that is considered as an input to the front end generator. The front end generator executes all syntactic and morphological processes that are necessary to produce the corresponding surface string. This involves: linearization, (i.e. constituent ordering), agreement handling, and inflection [21].

Simmons and Slocum developed a natural language system that generates sentences from a semantic network representation of knowledge, based on a case grammar [6]. The program generates surface structures from the augmented transition network (ATN). The object of the work was to substantiate the claim that "the semantic network adequately represents some important aspect of the meaning of discourse" [6].

Another generator, SHRDLU [6], contained several text generation devices that allow the system to answer questions. The basic techniques used for text-generation is "fill in the blank" and stored response patterns. For example, if an unfamiliar word was used, SHRDLU's response would be "I don't know the word...". Complex responses can be called by asking questions with "why" or "how" an action had been done. For "why", the system answered with "because <event>" or "in order to <event>" where <event> referred to a goal that the program had had when the action was taken.

Another generator [6] developed by Ross, employs a semantic net to represent the relations between words, which can be interpreted as their meaning. The task of this system was to compare two words, that is, find some semantic relation between them, and then to express the comparison in

"understandable, though not necessarily grammatically perfect, sentence".

Other programs and systems have been developed in this field. SEMTEX system [20], by Rosner, generates new stories from statistical data. GEOTEX system [16], by Khel, generates descriptive texts for geometric constructions. Yorick Wilks has developed a program that generates French [6] from a semantic base of templates.

1.4 RESEARCH IN ARABIC SENTENCE GENERATION

Development of a sentence generation system greatly depends on the language used. Most of the research for sentence generation has been related to many different languages but none to Arabic [7,17,18]. Thus, there is a little work in Arabic sentence generation. Only recently, with the trend of computer Arabization, research was geared towards Arabic. However, most of this research [1,2,4,15] was concentrated on the field of morphological and syntactic analysis.

With respect to morphological synthesis, a system developed by Hilal [13] was based on two methods of synthesizing a word. The first method (التوليد المعجمى), is to synthesize a word from the word root and its derivation. This was done in three stages :

1 - replace the (فاء) of feil of the derivation by the first letter of the root, the (ع) letter of the derivation by the second letter of the root, and the (لام) letter of the derivation by the third letter of the root,

2 - apply transformation rules that treat the special cases, such as (مصتّلح ---> مصطلح), and

3 - make a writing correction that handles the letter (همزه), such as (ااكل ---> أكل).

The second method (التوليد النصى) is to synthesize a word from a given preliminary word (root) and set of attributes:

- for a noun: its type (plural, dual, singular, masculine, and feminine), and its Irab cases (الحالات الأعرابيه) (nominative, accusative, and genitive),
- for a verb: its tense (past, present, and imperative), its irab cases (indicative, subjunctive, and jussive), and its type (plural, dual, singular, msculine, and feminine).

For generating nouns, the system includes a dictionary that shows the word root and type of its plural, all derivations of all the plural (جمع التكسير), a set of transformation rules, and transformation rules processing unit. Three steps are used to generate verbs, the first step is to use the proposed form al-nmadhj (النماذج) to get an intermediate word that can be the final word. Al-nmadhj is a set of frames ("Gowalbs"), each of which is associated with a derivative. The second step is to apply the transformation rules to treat the cases that have (نون التوكيد). The last step, is to make writing corrections for some of the cases, such as (أأكل --> أكل).

The disadvantage of the system is that the system requires storage for all roots, morphological patterns and standard forms.

With respect to sentence generation, research in [5] drew its theoretical framework from three sources: the transformational generative grammar proposed by Chomsky, the case grammar matrix model, and the Arabic grammar proposed by the early Arab grammarians in the eight century A.D. The

proposed method for structuring a sentence is produced either by base-generated rules or by transformational rules which generate various general and specific meanings.

CHAPTER II

ARABIC SENTENCE GENERATOR

2.1 INTRODUCTION

The proposed system is an Arabic sentence generator (ASG). It has been designed as an independent module so that it can be easily integrated in any related application. In addition, the adopted design is also extensible and maintainable in the sense that it can easily accommodate any future updates or modifications.

This chapter discusses the design of the Arabic sentence generator.

2.2 DESIGN

ASG involves synthesis of morphology, and syntax. As morphological constructs are affected by their syntactic roles and end-case grammar rules of Arabic, ASG must incorporate end-case rules of Arabic in the construction of words and sentences. ASG is, therefore, designed to consist of four main modules: preprocessor, morphological synthesis, end- case synthesis, and syntactic synthesis. The interaction between these modules is done by passing information from one module to another module as described in section 2.3.

Before we describe each of the above modules in detail, we describe the ASG preprocessor module.

2.2.1 Preprocessor Module

The ASG has an input module that transforms the representation of the given input into an acceptable form to the syntactic synthesizer. In general, the input is a logical form which is represented as a predicate at the implementation level. The name of the predicate specifies the relation between its parameters.

The input is assumed to have the following forms:

- verbal relations are represented by the predicate verb(X,Y,..), where X is a verb, and Y is a subject,
- nominal relations are represented by the predicate nom(X,Y,..), where X is a subject, and Y is a nominal predicate, and
- for each special-article relation, including prepositional articles, 'naskh' articles, interrogative articles ..etc, there is a corresponding predicate, as shown in Figure 2.1.

Since verbal relations have a verb as one of the parameters, then it can be mapped into a verbal sentence such that the first word is a verb, or it can be mapped into a nominal sentences such that the first word is the Mubtada and the second construct is a Khabar which is a verbal sentence. However, nominal relations only have nouns as parameters, thus will be mapped into nominal sentences. As a matter of style, we choose to map the verbal relations into verbal sentences.

/* sentences starting with verb */

verb(X1,X2,....Xn).

/* sentences starting with 'naskh' articles */

anna_n(X1,X2,.....Xn).

iinna_n(X1,X2,.....Xn).

lkn_n(X1,X2,.....Xn).

kaan_n(X1,X2,.....Xn)

laala_n(X1,X2,.....Xn).

leta_n(X1,X2,.....Xn).

/* sentences starting with 'jzm' noun */

mn_j(X1,X2,.....Xn).

ma_j(X1,X2,.....Xn).

mhma_j(X1,X2,.....Xn).

mta_j(X1,X2,.....Xn).

aena_j(X1,X2,.....Xn).

anna_j(X1,X2,.....Xn).

hethama_j(X1,X2,.....Xn).

kefma_j(X1,X2,.....Xn).

aee_j(X1,X2,.....Xn).

/* cluase that has prepositional article */

jr_mjror(X,Y).

/* sentences starting with 'nfey' articles */

la_n(X1,X2,.....Xn).

ma_n(X1,X2,.....Xn).

ln_n(X1,X2,.....Xn).

/* relations represent qassam sentences */

w_allah_q(X1,X2,.....Xn).

lam_q(X1,X2,.....Xn).

/* sentences starting with noun */

nom(X1,X2,.....Xn).

/* sentences starting with 'jzm' articles */

lma_j(X1,X2,.....Xn).

lm_j(X1,X2,.....Xn).

l_j(X1,X2,.....Xn).

.inn_j(X1,X2,.....Xn).

idma_j(X1,X2,.....Xn).

idma_j(X1,X2,.....Xn).

/* sentences starting with estfham article or estfham noun*/

hl_s(X1,X2,.....Xn).

a_s(X1,X2,.....Xn).

mta_s(X1,X2,.....Xn).

aena_s(X1,X2,.....Xn).

ana_s(X1,X2,.....Xn).

ayan_s(X1,X2,.....Xn).

ay_s(X1,X2,.....Xn).

km_s(X1,X2,.....Xn).

ma_s(X1,X2,.....Xn).

mndha_s(X1,X2,.....Xn).

mn_s(X1,X2,.....Xn).

matha_s(X1,X2,.....Xn).

matha_s(X1,X2,.....Xn).

/* sentences starting with 'afal al-naqssah' */

kan_f(X1,X2,.....Xn). sara_f(X1,X2,.....Xn).

assbaha_f(X1,X2,.....Xn).

dhala_f(X1,X2,.....Xn). lysa_f(X1,X2,.....Xn).

amsa_f(X1,X2,.....Xn).

bat_f(X1,X2,.....Xn). adha_f(X1,X2,.....Xn).

Figure 2.1 : List of Special Article Predicates

Verbal and nominal relations do not determine the exact classification of the corresponding sentence; however, it is determined later by the syntactic synthesizer, as discussed in section 2.2.3.

Every predicate has a set of constructs (sub-sentence, phrase, words) as parameters, each of which has a set of attributes (root, derivation, type, tense, infix), as shown in Figure 2.1.

The preprocessor module performs two functions on the input:

- it determines the type of the sentence, for example, if the name of the predicate is "verb" then the sentence type is verbal, however, if the name of the predicate is "nom" then the sentence type is nominal,
- restructuring the given input into a syntactic structure according to the rules of the syntactic synthesizer module.

The output of this module is:

1- the type of the sentence to the most possible classification,

2- the sequence of constructs that make up the sentence with the following corresponding information:

   - type of the construct (sub-sentence, phrase, noun, verb, article),
   - for a noun, the type of the noun (plural, dual, singular, masculine, feminine), its derivation, root, and the infix.
   - for a verb, the type of the verb (plural, dual, singular, masculine, feminine), the tense of the verb (past, present, future), its root, derivation, and the infix.

- for an article, the type of the article (interrogative, ...etc), pattern, and infix.
- for a sub-sentence, the type of the sub-sentence, and the sequence of the constructs that make up the sub-sentence with their information.
- for a phrase, the type of the phrase, and the constructs that make up the phrase with their information.

We used logical forms for meaning representation for the following features:

- recursive definitions can be used,
- facts and rules coexist in the description of any relationship,
- the expression of a problem in logic often corresponds to our intuitive understanding of the domain,
- logical assertions can be added to the database independently of the database "logic is modular",
- it has standard methods to determine the meaning.

2.2.2 Morphological Module

The function of the morphological module is to construct the word in its final shape. The construction of this module is based on linguistic principles used in the morphological analyzer module developed in [1] in order to take advantage of the work done earlier in this field, and to implement and integrate the synthesizer with NAUS, with minimum overhead.

The morphological analyzer consists of four submodules: control, verb, noun, and article submodules as shown in Figure 2.2. The functions of the control submodule are: determination of the type of the word (verb, noun, article), determination of the size of the word, and checking the word in the article module, noun module, or verb module (depending on the length of the word and the language properties).

The verb submodule is dedicated for the manipulation of morphological analysis of verbs. It determines the root, the derivation, tense of the verb (present, past), the type of the verb (plural, dual, singular, masculine, feminine), the external affixes and those letters that change the meaning of the word (prefix, infix or suffix).

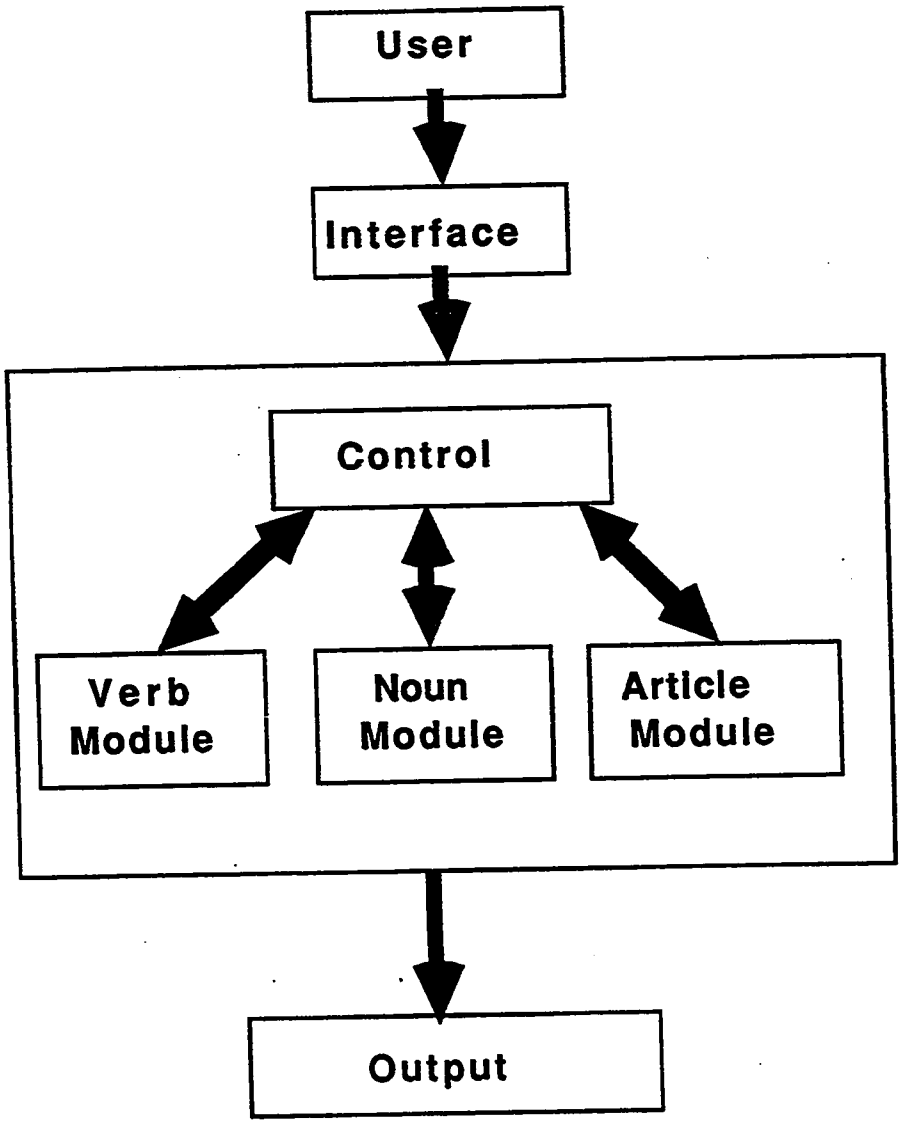

Figure 2.2 : Morphological Module Design

The noun submodule determines the root, the derivation, type of the noun (plural, dual, singular, masculine, feminine), the external affixes letters and those letters that change the meaning of the word (prefix, infix or suffix). It consists of a set of predicates that were developed to cover all cases of noun words of size two through fourteen letters.

Articles are limited and do not have infix letters. They are made up of two to seven letters. This module consists of a set of predicates that were developed to cover all these cases. In this module, the root, type of the article, prefix, and suffix of the article are determined.

Based on this design, the morphological synthesizer module consists of three sub-modules: noun, verb, and article. As the structure of the verb and noun modules is the same, we will first consider them concurrently and then the article submodule.

Verb and Noun Submodules

These submodules are designed as a set of predicates such that one or more can process one case of the verb or noun derivations. The verb and noun derivations are shown in Table 2.1, and Table 2.2 respectively. Every verb and noun is determined by the root of the word, its derivation, tense, type of the word (plural, single, dual, feminan, msculine), prefix and suffix. These properties are considered as conditions that must be satisfied by the input, in order for the predicate to process the right case. Implementation details are given in section 3.2.

Matching the input of the module against the conditions of all the predicates will invoke the right predicates to process the input. After a correct matching is done, the predicate will construct the word by attaching the suffix, the infix and prefix, if there, to the root of the word.

Article Submodule

The design of the article module has the same design principles of the verb and noun modules. It is designed as a set of predicates, each of which processes one case of the article patterns. Articles do not have derivations, however, they have a limited number of the patterns.

Every article pattern is determined by the type of the article, prefix and suffix. These properties are considered as conditions that must be satisfied by the input, in order for the predicate to process the right case. Implementation details are given in section 3.2.3.

TABLE 2.2: Noun Derivations

فاعل	فُعّال
مفعل	فعل
مفاعل	أفعل
منفعل	فعلاء
مفتعل	فعلان
متفاعل	فعلى
متفعّل	فعال
مفعل	فعول
مستفعل	فيعل
مفعلل	فعيل
متفعلل	مفعال
مفعول	مفعل
مفعل	مفعلة
مفتعل	فاعلة
مفعّل	فاعول
متفاعل	فعّالة
متفعّل	فعالة
مفعل	فعلان
مستفعل	فعال
مفعلل	فعيل
متفعلل	
فعيل	
فعوله	
فعل	
فعّال	
مفعال	
فعيل	

TABLE 2.1: Verb Derivations

أفعل
فعّل
فاعل
انفعل
افتعل
تفاعل
افعلّ
استفعل
افعوعل
افعالّ
افعوّل
فعلل
تفعلل
افعنلل
افعللّ
تفعّل
فعل
فيعل
فنعل

2.2.3 Syntactic Module

The purpose of this module is to build correct syntactic Arabic sentences. The input to this module is :

- a sequence of words.
- for each word in the sentence, a corresponding set of properties:
  - root and derivation,
  - prefix, infixes, and suffix,
  - type (noun, verb, or article),
  - for a noun, its type (plural, dual, singular, masculine, feminine,
  - for a verb: its tense (past, present, or imperative), and its type (plural, dual, or singular, and perfect or imperfect),
  - type of article (interrogative, prepositional).

Based on the adopted classification in [2], there are eight different types of Arabic sentences. Figure 2.3,. shows the classification which include types and subtypes. This classification was proposed by [2] to define the common properties that group Arabic sentence structures into different classes.

Based on this classification, the syntactic module consists of eight sub-modules:

- Non-Article Nominal Simple sub-module

This module is dedicated for the sentence that starts with a noun, does not contain any article, and does not contain any sentence.

- Non-Article Verbal Simple sub-module

This module is dedicated for the sentence that starts with a verb, does not contain any article, and does not contain any sentence.

- Article Nominal Simple sub-module

This module is dedicated for the sentence that starts with a noun or with a number of articles followed by a noun, contains at least one article, and does not contain any sentence.

- Article Verbal Simple sub-module

This module is dedicated for the sentence that starts with a verb or with a number of articles followed by a verb, contains at least one article, and does not contain any sentence.

- Non-Article Nominal Compound sub-module

This module is dedicated for the sentence that starts with a noun, does not contain any article, and contains two or more sentences.

- Non-Article Verbal Compound sub-module

This module is dedicated for the sentence that starts with a verb, does not contain any article and contains two or more sentences.

- Article Nominal Compound sub-module

This module is dedicated for the sentence that starts with a noun or with a number of articles followed by a noun, contains at least one article, and contains two or more sentences.

- Article Verbal Compound sub-module

This module is dedicated for the sentence that starts with a verb or with a number of article followed by a verb, contains at least one article, and contain two or more sentences.

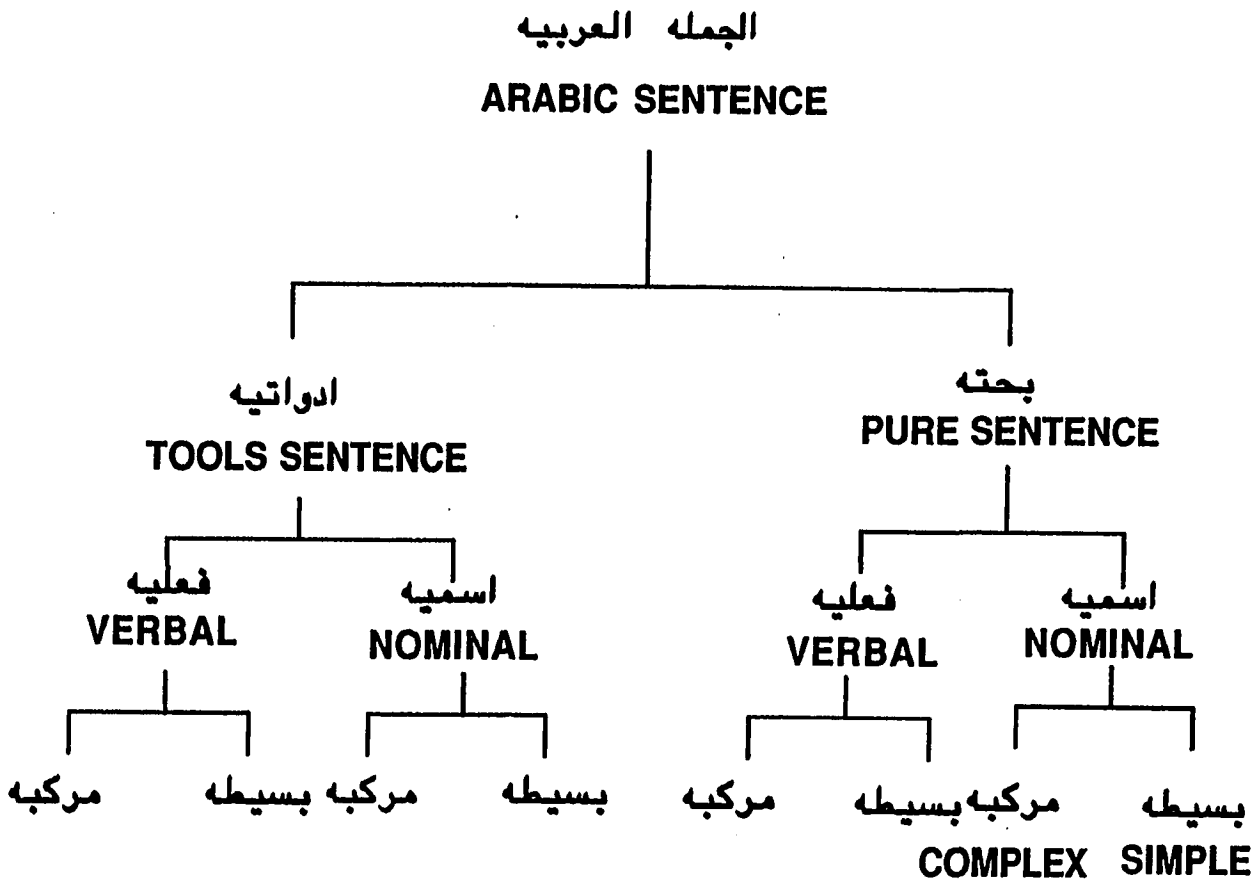

Figure 2.3 : Sentence Classifications

The syntactic module receives the input (from the preprocessor module) that is in an intermediate form, as discussed in section 2.2.1. Based on this input, the syntactic module initiates the right submodule to construct the sentence. In order for the syntactic module to construct a complete sentence, it must get some information from other modules (end-case, and morphological module). Passing this information back and forth, will create an interaction between the syntax, the morphological, and the end-case module. Section 2.3 discusses the interaction between these three modules in detail.

2.2.4 End-Case Module

The construction of this module will be based on the end case analyzer module that was developed in [4] in order to take advantage of the work done earlier in this field, and to implement and integrate the synthesizer with the NAUS, with minimum overhead.

The end-case analyzer [4] was subdivided into two main components: word-level end-case modules and general case module as shown in Figure 2.4. The word-level end-case module is also divided into three main modules: noun, verb, and article module as shown in Figure 2.4. Each one of these modules encodes the corresponding Arabic end-case grammar rules of all types of words as single words. These modules receive calls for end-case analysis of single words from the general case module which acts as an intermediate module, between the word-level end-case modules and the syntactic analyzer. The intermediate module receives these calls from the syntactic analyzer. These calls include either a simple word or a compound word. A compound word is a list of words, where the first word is the only one for which the end-case position is known and the other words in the list are qualifiers (توابع).

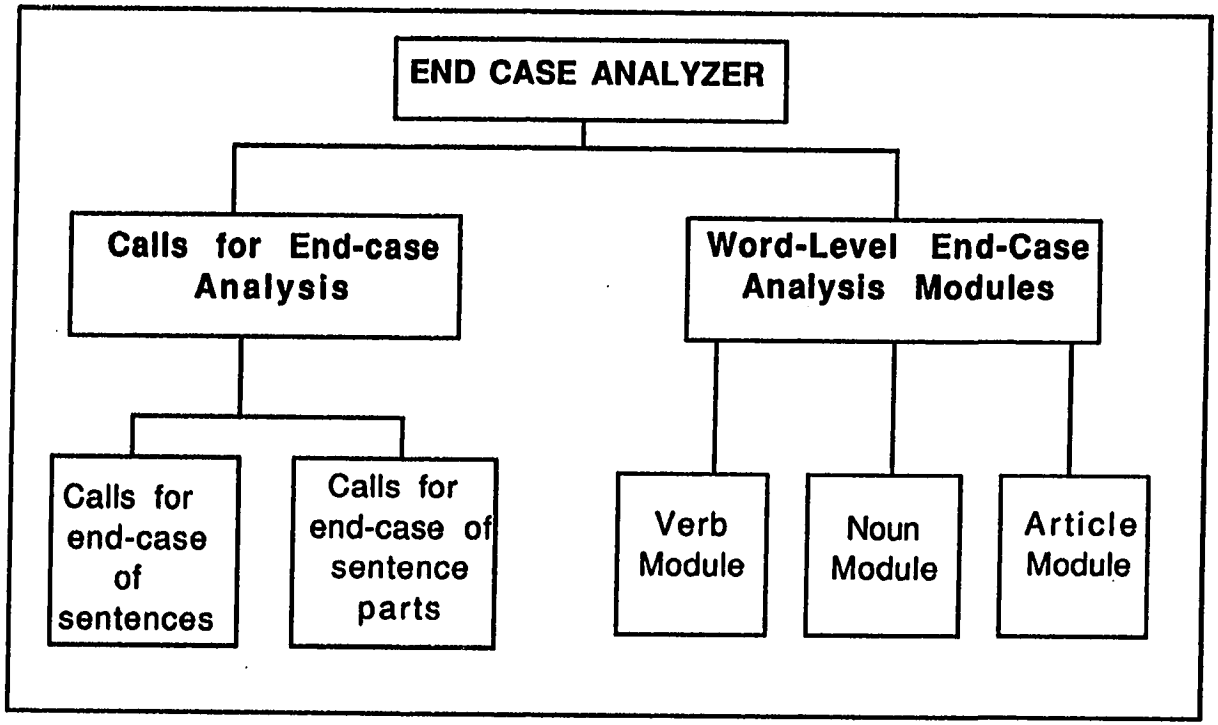

Figure 2.4: Design Structure of The End-Case Module

End-case analysis of a sentence as a whole unit is done according to a set of rules in Arabic end-case grammar. The sentence module encodes these Arabic end-case grammar rules of all types of sentences. From the end-case point of view, sentences in Arabic are divided into two main classes: sentences that have an end-case position, and sentences that do not have an end-case position. Table 2.3 shows these two classes and sentence types of each class. This module has no effect on the construction of sentences in the ASG, since the implicit end-case of a sentence does not affect the construction of the words.

The noun module consists of three main submodules: nominative, accusative, and genitive module, which are dedicated for the manipulation of end-case analysis of nouns that are in the nominative, accusative and genitive cases respectively. Each module encodes the corresponding Arabic end-case grammar rules of all types of nouns as single words assuming that they are in the corresponding case.

The verb module consists of three main components: imperative tense module, past tense module, and present tense module. Each module encodes the corresponding Arabic end-case grammar rules of the type of verb that the verb belongs to as single words.

The article module processes an article according to its identity. All the articles in Arabic are of the invariable type of words, that is, their terminal diacritics do not experience any change.

جدول تصنيف الجمل بحسب المحل من الاعراب

TABLE 2.3 : Sentence End-Case Classification

الجمل التى لها محل من الاعراب Sentences with an end-case place	الجمل التى لا محل لها من الاعراب Sentences with no end-case place
الجمله الواقعه خبر	الجمله الابتدائيه
الجمله الواقعه مفعولا به	الجمله المعترضه
الجمله الواقعه مضافا اليه	الجمله التفسيريه
الجمله الواقعه جوابا لشرط جازم	جملة جواب الشرط غير الجازم
الجمله الواقعه حالا	جملة جواب القسم
الجمله الواقعه صفه	جملة صلة الاسم الموصول
الجمله التابعه لجمله لها محل	الجملة التابعه لجملة لا محل لها

For each class of the Arabic words (nouns, and verbs) there are some end-case rules that determine its end-case characteristics, such as end-case position, (الموقع الأعرابى), and end-case mark, (العلامه الأعرابيه), according to its type.

The ASG end-case synthesis submodules are:

Noun Submodule

In [4], nouns were divided into two categories: invariable, and variable nouns, and each category has different types of nouns. Table 2.4 shows these two categories and noun types in each category.

An invariable noun is a noun that does not suffer any change in its end-case mark even when its end-case position in the sentence is changed. This type of noun has no effect on the structure of the word, since its end-case mark does not change.

A variable noun suffers changes in its end-case mark when its Irab case in the sentence changes. Some types of the variable nouns have the end-case mark as a diacritic, while for others it is a letter. Table 2.5 shows a list of these noun types and the corresponding end-case marks. The variable nouns that have their end-case mark as a diacritic have no effect on the word structure, since their end-case mark is not a letter. However, the undesignated noun that is in the accusative case can have a letter (الالف) , e.g. "فلاحا ", as its suffix, even when its end-case mark is a diacritic. The handling of this case is explained next in the Noun Submodule Design.

جدول الاسماء المعربه والمبنيه

TABLE 2.4 : Variable and Invariable Nouns

الاسماء المعربه Variable Nouns	الاسماء المبنيه Invariable Nouns
- معرب بالحركات : الضمه، الفتحه، الكسره × الاسم المفرد × جمع التكسير × جمع المؤنث السالم -معرب بالحروف: الواو، الألف، الياء × جمع المذكر السالم × المثنى × الأسماء الخمسه	-مبنيه بناء لازما × أسماء الاشاره × الضمائر × أسماء الشرط و أسماء الاستفهام × الأسماء الموصوله × أسماء الأفعال والأصوات × بعض الكنايات والظروف - مبنيه بناء عارضا × المنادى المعين × الأعداد المركبه × الاسم العلم المركب مزجيا × اسم لا النافيه للجنس

جدول الأسماء المعربة وعلامة اعرابها

TABLE 2.5 : Variable Nouns and their End-Case Marks

End-Case Mark علامة اعرابها			Noun Type نوع الاسم
الجر Genetive	النصب Accusative	الرفع Nominative	الحاله الاعرابيه
			- معرب بالحركات : الضمه، الفتحه، الكسره
الكسره الفتحه	الفتحه	الضمه	x الاسم المفرد
الكسره	الفتحه	الضمه	x جمع التكسير
الكسره	الكسره	الضمه	x جمع المؤنث السالم
			-معرب بالحروف: الواو، الألف، الياء
الياء	الياء	الواو	x جمع المذكر السالم
الياء	الياء	الالف	x المثنى
الياء	الالف	الواو	x الأسماء الخمسه

Noun Submodule Design

Since the invariable nouns and variable nouns that have diacritics as their end-case mark have no effect on the word structure, then they don't contribute to the suffix, and the prefix of the word. However, an exceptional case (the undesignated noun) can affect the structure of the word, as mentioned before.

Nouns with letter-end-case marks have three Irab cases (الحالات الاعرابيه): nominative, accusative, and genitive. A noun can be in one of these cases according to the end case position (الموقع الاعرابي) that it occupies in a sentence. For a complete list of these Irab cases and the possible end- case positions that cause a noun to be in those Irab cases see Table 2.6. Table 2.7 is extracted from [4] to show the different types of nouns and the corresponding letter-end-case marks associated with each type.

At the single-word level, the noun module consists of three main parts:

a- The nominative noun module: this module is designed as a set of rules that encodes the corresponding Arabic end-case grammar rules of all types of nouns that are in nominative case. Table 2.7 shows the different types of nouns and the corresponding end-case marks associated with each type.

b- The accusative noun module : this module is dedicated for the manipulation of end-case synthesis of nouns assuming that they are in the accusative case. Table 2.7 shows the different types of nouns and the corresponding end-case marks associated with them in the accusative

case. For the case of the undesignated noun, there are two rules that handle it. The first rule handles the case where the undesignated noun is (مضاف), and the second rule handles the general case where the noun has a suffix (الف).

c- The genitive noun module: this module is designed as a set of rules that encodes the corresponding Arabic end-case grammar rules of all types of nouns that are in genitive case. Table 2.7 shows the different types of nouns and the corresponding end-case marks associated with them in the genitive case.

جدول الحالات الاعرابيه للأسماء مع المواقع الاعرابيه المسببه لها

TABLE 2.6 : Noun Irab Cases and the Corresponding End-Case Positions

الحاله ← الرفع Nominative	النصب Accusative	الجر Genetive
الموقع الاعرابى للكلمه Word End-Case Position		
مبتدأ	اسم ان ولا النافيه	مضاف اليه
خبر	خبر كان	مجرور بحرف جر
اسم كان	مفعول به	تابع لمجرور
خبر إن	مفعول معه	
خبر لا النافيه للجنس	مفعول فيه	
فاعل	مفعول مطلق	
نائب فاعل	حال	
منادى علم	تمييز	
منادى نكره مقصود	منادى نكره	
تابع لمرفوع	تابع لمنصوب	

جدول الأسماء المعربة وعلامة اعرابها

TABLE 2.7 : Variable Nouns and their End-Case Marks

نوع الاسم Noun Type	علامة اعرابها End-Case Mark		
الحاله الاعرابيه	الرفع Nominative	النصب Accusative	الجر Genetive
-معرب بالحروف: الواو، الألف، الياء			
x جمع المذكر السالم	الواو	الياء	الياء
x المثنى	الالف	الياء	الياء
x الأسماء الخمسه	الواو	الالف	الياء

The end-case module receives calls that include a simple or a compound noun in their arguments for end-case synthesis of nouns from the syntactic synthesizer. A compound noun is a list of nouns, and the first noun is the only one for which the end-case position is known, but the other nouns are qualifiers (توابع). For this reason we built an intermediate module between the word-level end-case modules and the syntactic synthesis module. The general case module simplifies the calls for end-case synthesis of a sequence of nouns into simple calls for end-case synthesis of single words.

Verb Submodule

Figure 2.5 shows the different types and classes of verbs in Arabic. However, from the end-case point of view verbs fall into one of the two categories: invariable verbs, and variable verbs. Figure 2.6 illustrates these two categories.

An invariable verb does not suffer any change in its end case mark even if it was preceded by any of the verbal articles that affect the end-case of a verb. These invariable verbs are: all the past tense verbs, all the imperative verbs, and the present tense verbs that are suffixed with (نون التوكيد), or (نون النسوة).

A variable verb suffers from changes in its end-case mark according to changes in its Irab case in the sentence. Variable verbs have three Irab cases (Figure 2.7): indicative (مرفوع), subjunctive (منصوب), or jussive (مجزوم). A verb is said to be in the subjunctive case if it is preceded by one of the accusative articles (حروف النصب). If it is preceded by one of the jussive

articles, (حروف الجزم), then it is said to be in the jussive case. Otherwise, it is said to be in the indicative case.

The end-case mark for a verb depends on its case, its type, and its terminal letter. That is, for some verb types the end-case mark is a diacritic, while for others it is a letter.

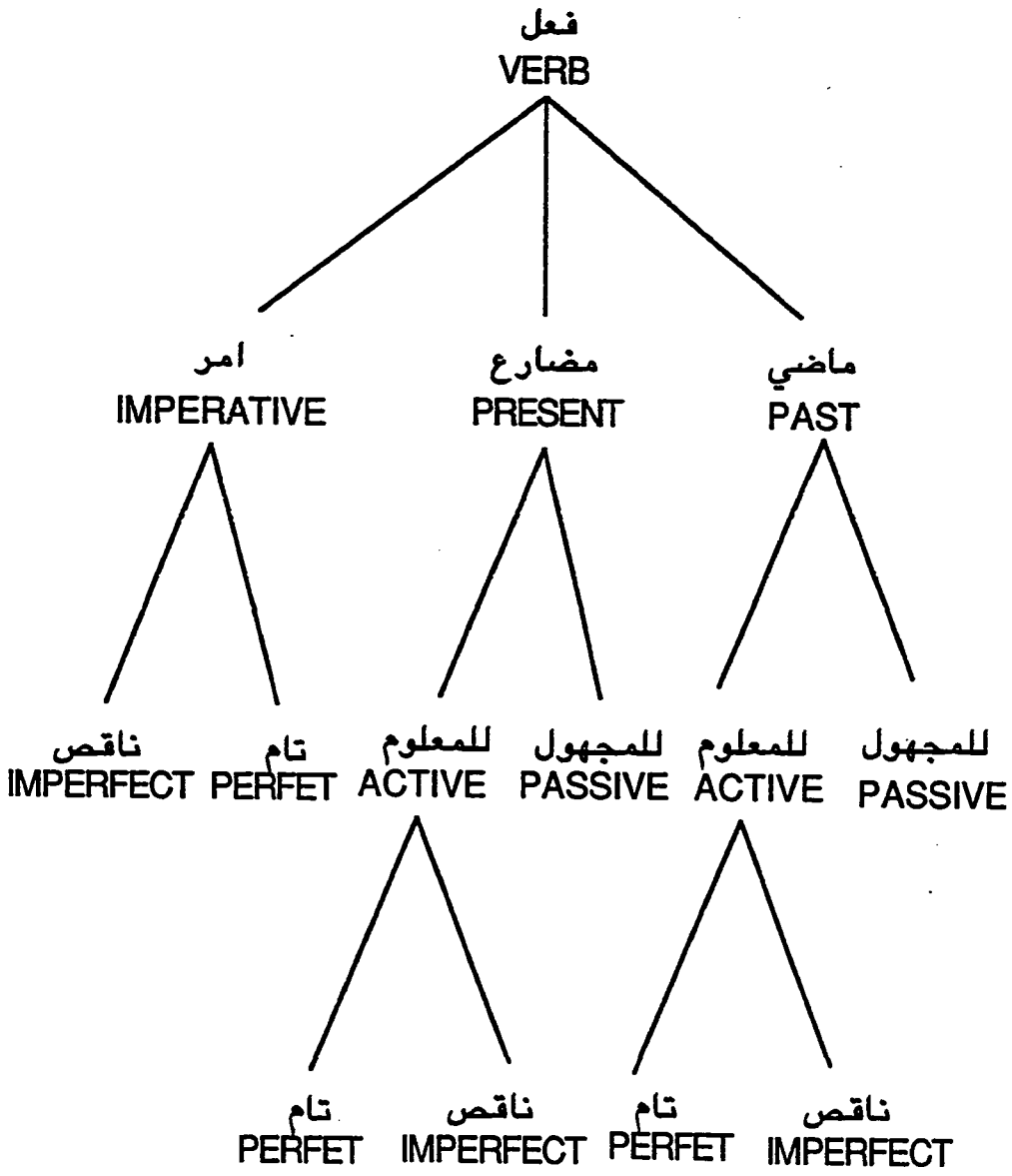

Figure 2.5 : Verb Types

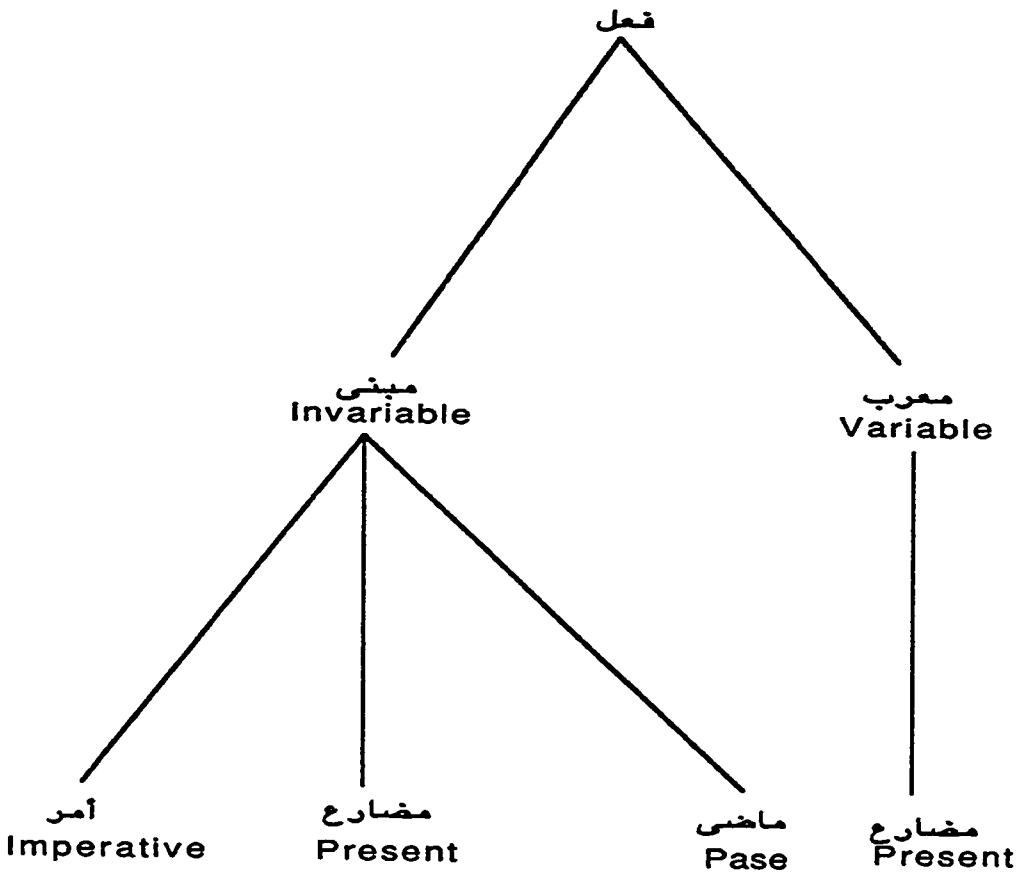

Figure 2.6 : Verb Irab Categories

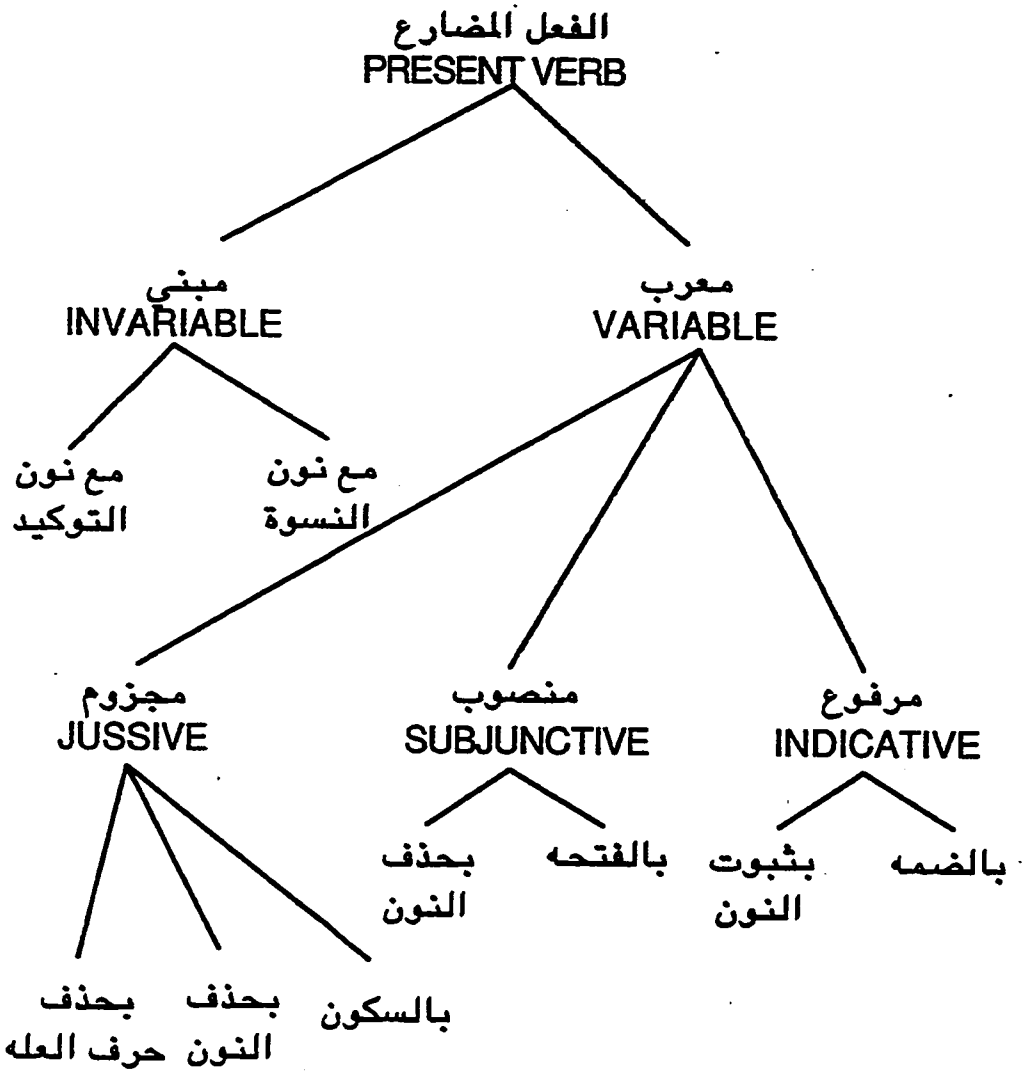

Figure 2.7 : Present Verb Irab Categories

Verb Submodule Design

The verb module consists of the following three main parts: imperative tense module, past tense module, and present tense module. The imperative verbs have two kinds: perfect and imperfect, and both of them are processed in the same way except for determining the verb's type. Accordingly, only one module, with a parameter for the verb's type, suffices to process all the imperative tense verbs.

Past tense verbs are classified into two subclasses: active voice verbs, and passive voice verbs. Accordingly, we have two modules, one for the active voice and the other for passive voice.

The present tense verbs are classified into two subclasses: active voice, and passive voice. From the end-case point of view, present tense class is subdivided into variable and invariable verbs. Variable verbs are further divided into three subdivisions: indicative, subjunctive, and jussive. Applying this division to the two classes of the present tense, namely, active voice and passive voice classes, we have eight modules in total.

Each of the above modules, encodes the corresponding Arabic end-case grammar rules for the corresponding type of verb, and these modules receive and process calls that are initiated from the syntactic synthesizer, for end-case synthesis of single verbs.

Verbs with letter-end-case marks have three Irab cases: indicative, subjunctive, and jussive. A verb can be in one of these cases according to the

end-case position that it occupies in a sentence. Table 2.8 shows the list of these Irab cases (letter-end-case mark) and the corresponding end-case positions.

The third-letter-vowel verb is processed in the morphological module since it requires omitting the vowel letter in the jussive case.

جدول الأفعال وعلامة إعرابها

TABLE 2.8 : Verb Irab Cases & the Corresponding End-Case Positions

End-Case Mark علامة إعرابها			Noun Type نوع الفعل
الجزم JUSSIVE	النصب SUBJUNCTIVE	الرفع INDICATIVE	الحاله الإعرابيه
حذف النون (وا ، ا ، ي) حذف حرف العلة	حذف النون (و\ ، ا ، ي)	ثبوت النون (ون ، ان ، ين)	الأفعال الخمسة الفعل المعتل الآخر

Article Submodule

Figure 2.8 shows the different classes of articles in Arabic. These classes are: active articles that affect the succeeding words, and non-active or idle (غير عامله) articles that have no effect on the following word. The active articles change the succeeding word's case. This change depends on the article's type and the type of the affected word as well.

The active articles have three classes (as shown in Figure 2.8): nominal (applicable for nouns only), verbal (applicable for verbs only), and nominal-verbal(applicalble for nouns or verbs). The nominal class includes, prepositional articles, (حروف الجر) that put the succeeding noun in the genitive case, and 'naskh' articles, (حروف النسخ), which put the succeeding noun in the accusative case, ... etc.

The verbal articles are divided into two main classes (as shown in Figure 2.8): articles that put present verbs in the subjunctive case, and articles that put present verbs in the jussive case. The verbal articles only precede a verb and affect its case.

An example of the nominal-verbal articles that may precede verbs or nouns is the set of articles known as conjunction articles (حروف العطف). All the articles are of the invariable type of words, that is, an article terminal diacritic does not suffer any change wherever it occurs in a sentence.

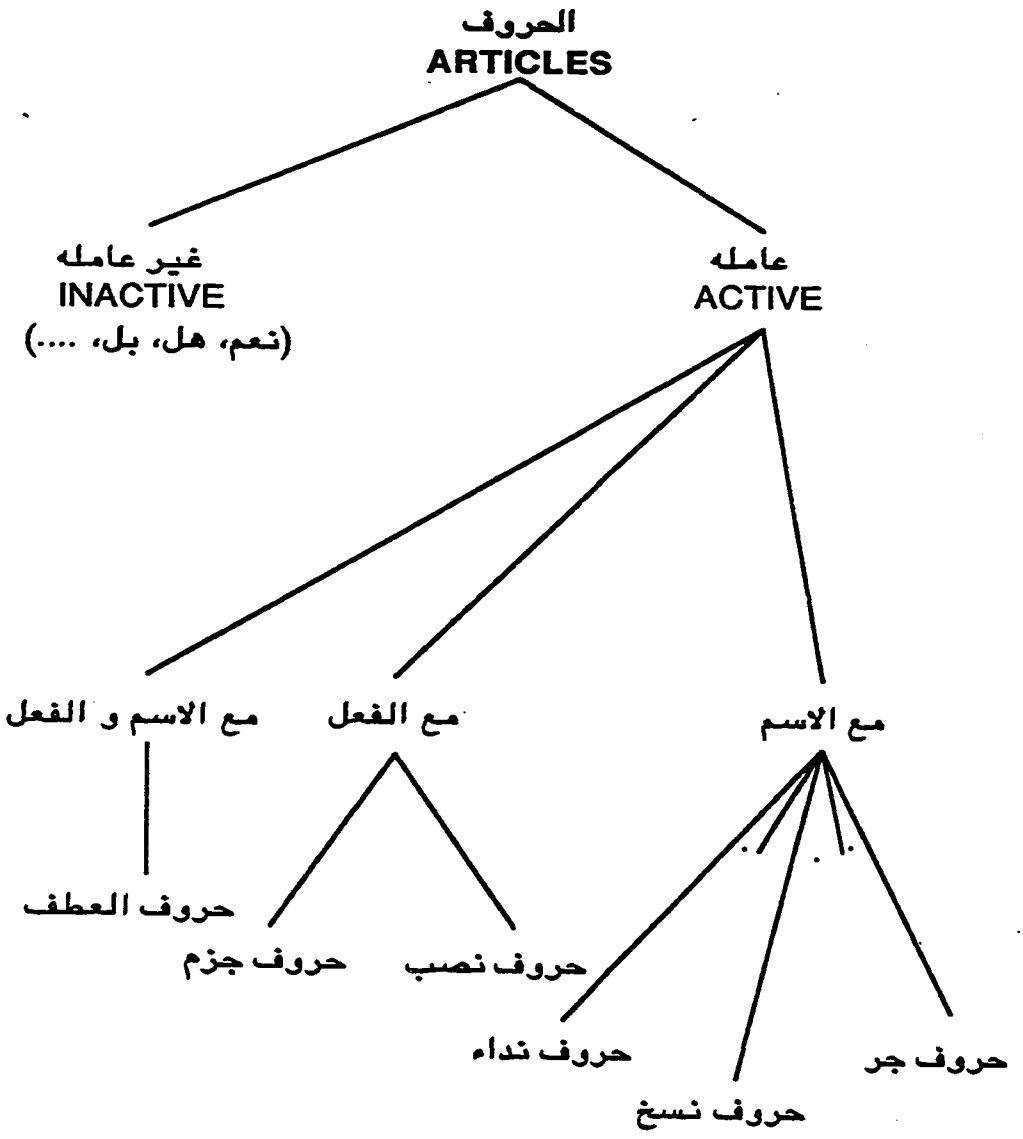

Figure 2.8 : Article Types

Article Submodule Design

The article module (Figure 2.4) was designed to process any kind of article according to its identity. The input to this module is:

- type of article (interrogative, prepositional,..etc),
- root, and
- its pattern.

The output of this module is the article itself,

2.3 The Interaction Between The Modules

In order to construct a complete correct sentence, interaction between the syntactic, end-case, and morphological modules is necessary. In ASG, the syntactic module controls a proper sequence of interactions with other modules.

Construction of a sentence is broken down by the syntactic module into the construction of each individual element (words, phrases, or sentences) of the sentence. If the element is a phrase or a sentence then the syntactic module will break it down into individual words, and then for each element (word) in the sentence, the syntactic module passes the element along with its attribute list and its end- case position to the end-case module in order to determine its end-case marks (العلامه الأعرابيه), which could be either a diacritic or a letter. For the case where the element is compound, the syntactic module passes the element to the general case module, which then passes each word of the element along with its attribute list to the end-case module to determine its end-case marks.

In case the element has a letter as an end-case mark, the end-case synthesizer determines the letter as specified in section 2.2.4. This end-case mark obtained from the end- case module is added as an affix to the list of attributes of the element and is sent back to the syntactic or general module. The syntactic module forwards the element along with its new compiled attribute list to the morphological module. The morphological module uses the given information to construct the word in its final shape and returns the constructed word to the syntactic module.

The syntactic module forwards the element along with its new compiled attribute list to the end-case analyzer to check the correctness of the constructed word. If the construction of the word is not proper, then the end-case will reject it, and send it back to the syntactic module. The syntactic module will call the morphological module again to choose another option. This cycle is repeated until the correct option is chosen.

2.4 EMBEDDING THE ARABIC SENTENCE GENERATOR INTO NAUS

The integration of ASG into NAUS involves the determination of the interactions of ASG with other modules of NAUS. The input to ASG is a sentence meaning that comes from the semantic module in NAUS, and the output is a complete correct sentence that is sent to the interface module in NAUS.

The Arabic Sentence Generator (ASG) consists of four modules: preprocessor module, morphological module, end-case module, and syntactic module. Since we need to build morphological, syntactic, and end-case synthesizer, we have two options:

1- To develop the morphological, syntactic, and end-case synthesis modules independent of the corresponding analyzers, or

2- To embed the corresponding synthesizers into the morphological, ssyntactic, and end-case analyzer modules, respectively.

We adopt the second approach to minimize the overhead in the implementation of ASG, and to take advantage of the already developed modules of NAUS. Furthermore, taking advantage of the assoctivity of prolog (declarative languages) with respect to unification, minimizes program size and development effort.

To integrate the synthesizers with the analyzers, a suitable strategy has to be followed. The strategy, which we have followed is:

- to distinguish between the analysis mode and synthesis mode. In analysis mode, the word and the word length will be given as

an input to the module, however, in the synthesis mode the word will not be given as input. From this fact we can distinguish between the analysis and synthesis by checking the existence of the word.

- to keep track of the information that is given in analysis but not in the synthesis, and vice versa.
  Some variables will be given in analysis mode but not in the synthesis mode, and these variables must be detected and treated differently in the synthesis mode.

Preprocessor

The preprocessor is embeded into NAUS as an independent module (which is currently called by the user) to generate a sentence from a given representation. The preprocessor calls the syntactic module after preparing the input in the proper shape

Morphology

In the morphological analyzer module [1], Arabic words are classified into three classes: verbs, nouns and articles. Verbs and nouns can be derived in many different forms using the bare roots of the words and a set of letters, "affixes" as suffixes, prefixes and infixes to the roots.

To integrate the morphological synthesis into the analyzer, we had to augment the sub-modules (verb, noun, article) in the morphological analyzer module.

The verb submodule

The input to the verb analysis submodule is the word itself and its length and the output is :

- type of word (noun, verb, article),
- type of verb (plural, dual, singular, masculine, feminine),
- tense of verb (past, present, future),
- root, and derivation, and
- type and identity of affixes (prefix, infix, suffix).

The modified version is to perform two functions : analysis and synthesis. In synthesis, the output of the module is the word (and its length), and the input is :

- type of word (noun, verb, article),
- type of verb (plural, dual, singular, masculine, feminine),
- tense of verb (past, present, future),
- root, and derivation, and
- infix, and
- prefixes and suffixes are sometimes given and sometimes are not, as explained earlier in section 2.3.

The noun submodule

The input to the noun analysis submodule is the word and its length, and the output is :

- type of word (noun, verb, article),
- type of noun (plural, dual, singular, masculine, feminine),
- root, and derivation, and

- type and identity of affixes (prefix, infix, suffixes)

The modified version is to perform two functions : analysis and synthesis. In synthesis, the output of the module is the word (and its length), and the input is :

- type of word (noun, verb, article),
- type of noun (plural, dual, singular, masculine, feminine),
- root, and derivation, and
- infex, and
- prefixes and suffixes are sometimes given and sometimes are not, as explained earlier in section 2.3.

The article submodule

The input to the article analysis submodule is the word, and the output is :

- type of word (noun, verb, article),
- type of article (interrogative, prepositional,..etc),
- root,
- type and identity of affixes (prefix, suffixes).

The modified version is to perform two functions : analysis and synthesis. In synthesis, the output of the module is the word, and the input is :

- type of word (noun, verb, article),
- type of article (interrogative, prepositional,..etc),
- root.

Syntax

The input to the syntactic analyzer is a sentence to be checked for syntactic correctness, and the output is the correctness of the input sentence, grammatical properties of its constituents, and the sentence syntax [2,15]. The program mainly consists of eight modules. Each module is dedicated to implement one type of the Arabic sentence based on the adopted classification. However, in the augmented version, we will have the same eight modules performing sentence analysis and synthesis. In sentence synthesis the input is a list of words with their attributes, and the output is a complete correct sentence.

With respect of the strategy of integrating synthesis with analysis, we followed the same strategy that was mentioned before, in addition to the ordering of the calls to the end-case module. The calls to the end-case module are made before the calls to the morphological module. The origional calls for the end-case analyzer are used for verification only.

End-case

The input to the end-case analyzer is:

- the word,
- end-case position of the word,
- root, derivation of the word,
- type of the word (noun, verb, or article),
- type of noun (plural, dual, singular, masculine, feminine),
- type of verb (plural, dual, singular, masculine, feminine),
- tense of the verb, and
- type of article(interrogative, prespositional, ..etc).

The output of the end-case analyzer is the end-case mark of the word.

The modified version is to perform two functions: analysis and synthesis. In synthesis, the output of the module is the suffixes and prefixes, and the input is:

- end-case position of the word
- root, derivation of the word
- type of the word (noun, verb, or article)
- type of noun (plural, dual, singular, masculine, feminine),
- type of verb (plural, dual, singular, masculine, feminine),
- tense of the verb,
- type of article(interrogative, prespositional, ..etc).

The following chapter shows the implementation details implied by the integration policy presented in this chapter.

CHAPTER III

IMPLEMENTATION OF ASG

ASG was implemented as a set of modules, each of which is composed of a set of rules. The ASG was implemented using Arity Prolog interpreter.

The program mainly consists of four modules: preprocessor, morphological, syntactic, and end-case synthesizers. The implementation of these modules are discussed in turn in the following sections.

3.1 PREPROCESSOR

The preprocessor was implemented as a set of predicates. Some of these predicates change the structure of the input, and some of them produce the type of the constructs and the type of the given sentence. Appendix (A) shows some predicates of the preprocessor.

3.2 MORPHOLOGY

To embed the morphological synthesis into the morphological analyzer, we had to augment the sub-modules (verb, noun, article) in the morphological module.

3.2.1 The Verb Submodule

To perform the changes we had to modify most of the predicates, and to add new predicates. One important point about the modification is that the modified submodules must work for both the analysis and synthesis modes.

The changes are :

1- The predicate ***concat(A,B,C)*** is used to assign to C the concatenation of A, and B. However, in synthesis we need a predicate that performs this function and another function which, when given C and A or B can find the value of the third variable. For this reason we built a predicate ***cocna(A,B,C)***, as shown below, that performs these two functions.

CONCA(A,B,C):- *concat(A,B,C); word_lst(C,[A,B]).*

The function of the predicate ***word_lst***(C,[A,B]) is to breake a word represented by the variable C into two parts A and B.

We replaced some of the predicates ***concat(_)*** by the predicate ***conca(_)*** where necessary, since ***conca*** can perform the function of ***concat*** and the other function.

As an example of the replacement of ***concat*** by ***conca***, the predicate before the replacement takes place is:

vpre_test4([A,B,C,D],RO,DE,TY,PR,"",""):-

member(D,["ت","ي","ن","ا","أ"]), concat([A,B,C],RO),

concat("فعل ",D],DE), TY="مضارع ", PR=D.

The predicate after the replacement takes place is:

vpre_test4([A,B,C,D],RO,DE,TY,PR,"",""):-

member(D,["ت","ي","ن","ا","أ"]), conca([A,B,C],RO),

conca("فعل ",D],DE), TY="مضارع ", PR=D.

2- Delaying the call to the predicate ***conca(E,F,P)*** in some predicates if the variables E, F, and P are all unbound variables in synthesis mode. For example, in the following predicate

```
vpre_test6([A,B,C,D,E,F],RO,DE,TY,PR,"","");-
    concat(E,F,P),
    member(P,["ول","فل","وس","فس"]),
    member(D,["ت","ي","ن","ا","أ"]),
    concat([A,B,C],RO), concat(["فعل ",D,P],DE),
    concat([D,P],PR), TY="مضارع".
```

E, and F are bound, and P is unbound in the analysis mode. However, in synthesis mode E, F, and P are all unbound. In this case we have to bind the variable P first, and then we make the call to the predicate ***conca***. Binding the variable P can be done by delaying the call to the predicate ***conca*** after the predicate ***member***. After performing the change, the predicate is as listed below

```
vpre_test6([A,B,C,D,E,F],RO,DE,TY,PR,"","");-
    member(P,["ول","فل","وس","فس"]),
    member(D,["ت","ي","ن","ا","أ"]), conca(E,F,P),
    conca([A,B,C],RO), DE="فعل " conca([D,P],PR), TY="مضارع".
```

In the analysis mode, the predicate ***member(P,["ول","فل","وس","فس"])*** checks if the variable P is part of the list , ["ول","فل","وس","فس"] or not. While in synthesis, it assigns to the variable P one of the values that is in the list ["ول","فل","وس","فس"]. This modification costs the analysis operation to consume more time by backtracking every time to select the right variable.

3- Binding some variables in some predicates before using them in the synthesis mode. For example, in the following predicate

```
verb_test5([A,B,C,D,E],T):-
    vpre_test5([A,B,C,D,E],RO,DE,Ty,Agender,Number,PR,IN,SU),
    concat([A,B,C,D,E],Oword),
    T=[Oword,R,DE,Ty,Agender,Number,PR,IN,SU].
```

The word [A,B,C,D,E] and the attributes RO, DE, Ty, PR, IN, and SU are unbound in the synthesis mode. To bind the variables RO, DE, Ty, PR, SU and IN we replace the variable T by its value in the first step; the predicate is as shown below:

```
verb_test5([A,B,C,D,E],
    [Oword,R,DE,Ty,Agender,Number,PR,IN,SU]):-
    vpre_test5([A,B,C,D,E],RO,DE,Ty,PR,IN,SU),
    concat([A,B,C,D,E],Oword),
```

One of the input parameters is the derivation of the word. For a correct operation we change the derivation for every word to be in the past tense third singular form. This modification has nothing to do with the synthesis but is necessary for correct operation of the module.

3.2.2 The Noun Submodule

The changes made in this sub-module are:

1- Replacing all the predicates ***concat(_)*** by the predicate ***conca(_)***, for the same purpose as mentioned above in section 3.1.1.

AS an example of the replacement of ***concat*** by ***conca***,

the predicate before the replacement takes place is:

npre_test4([A,B,C,D],RO,DE," مفرد "," ",D,"",""):-

member(D,["ا" ، "ا" ، "م" ، "ل" ، "ب" ، "ك" ، "ف" ، "و"]),

concat([A,B,C],RO), concat(" فعل ",D],DE).

The predicate after the replacement takes place is:

npre_test4([A,B,C,D],RO,DE," مفرد "," ",D,"",""):-

member(D,["ا" ، "ا" ، "م" ، "ل" ، "ب" ، "ك" ، "ف" ، "و"]),

conca([A,B,C],RO),

conca(" فعل ",D],DE).

2- Delaying the call to the predicate ***conca(E,F,P)*** in som predicates, for the same purpose as mentioned above in 3.1.1.

For example, the predicate before performing the change is,

npre_test7([A,B,C,D,E,F,G],RO,DE,"مفرد ","","",PR,IN,SU):-

concat(F,G,M),

member(M,["ال" ، "ول" ، "قل" ، "لا" ، "قب" ، "وك" ، "فك"]),

npre_test5([A,B,C,D,E],RO,DEE,SDP,MF,PRE,IN,SU),

concat(PRE,M,PR),

concat(DEE,M,DE).

After the modification, it is changed to

npre_test7([A,B,C,D,E,F,G],RO,DE,"مفرد ","","",PR,IN,SU):-

member(M,["ال" ، "ول" ، "قل" ، "لا" ، "قب" ، "وك" ، "فك"]),

conca(F,G,M),

npre_test5([A,B,C,D,E],RO,DE,SDP,MF,PRE,IN,SU),

concat(PRE,M,PR).

3- Binding some variables in some predicates before using them in the synthesis mode. For example, in the following predicate

noun_test5([A,B,C,D,E],T):-

npre_test5([A,B,C,D,E],RO,DE,SDP,MF,PR,IN,SU),

concat([A,B,C,D,E],Oword), T=[Oword,R,DE,SDP,MF,PR,IN,SU].

The word [A,B,C,D,E] and the attributes RO, DE, Ty, PR, IN, and SU are unbound in the synthesis mode. After performing the change, it becomes

noun_test5([A,B,C,D,E],[Oword,RO,DE,SDP,MF,PR,IN,SU]):-

npre_test5([A,B,C,D,E],RO,DE,SDP,MF,PR,IN,SU),

concat([A,B,C,D,E],Oword).

3.2.3 The Article Submodule

The changes that made in this sub-module are:

1- Replacing all the predicates **concat** by the predicate **conca**, as mentioned before in the verb, and noun submodules.

As an example of the replacement of **concat** by **conca**, the predicate before the replacement takes place is:

find_art2([A,B],T,Root,Type,B,""):-

concat([A,B],Root),

member(Root,["ال","أي"]),

Type = " حرف تفسير", T=2.

After the modification it is changed to

find_art2([A,B],T,Root,Type,B,""):-

conca([A,B],Root),

member(Root,[" أن"," أي"]),

Type = " حرف تفسير", T=2.

2- Binding the variables RO, Type, TC, PR, and SU by replacing the variable T by its value in the first step.

The predicate before the modification is:

art_test2([A,B],T):-

find_art2([A,B],TC,RO,Type,PR,SU),

concat([A,B],Oword),

T=[Oword,TC,RO,Type,PR,SU].

After the modification, it is changed to

art_test2([A,B],[Oword,TC,RO,Type,PR,SU]):-

find_art2([A,B],TC,RO,Type,PR,SU), concat([A,B],Oword).

3.3 END-CASE

To embed the end-case synthesis into the end-case analyzer, we had to augment the sub-modules (verb, noun, and article) in the end-case analyzer module.

3.3.1 The Noun, Verb and the Article Submodules

To perform the changes we had to modify most of the predicates in the analyzer, and to add new predicates. The augmented submodule works for both analysis and synthesis modes.

There is one common change that has been made for all the submodules. The change is performed by avoiding the predicate ***substring(X,a,b,PART)*** during the synthesis mode. To resolve this problem we used the if-then statement to distinguish between the analysis and synthesis mode by testing the existence of the variable X, which is boundduring the analysis mode, but not during the synthesis mode. The function of this predicate is to test if PART is a substring of X. For example, the following predicate

```
esm_marfo(X0,[X1,_,X3,$اسم$,$مؤنث$,$جمع$,_,_,_,Part],Out):-
  substring(X1,0,2,Part) ,   Part == $ات$,
  Out = $جمع مؤنث سالم مرفوع بالضمه الظاهره على اخرها$,
  rite([Out,' : ',X0,' : ',X1]),nl.
```

is modified to

```
esm_marfo(X0,[X1,_,X3,$اسم$,$مؤنث$,$جمع$,_,_,_,Part]  ,Out):-
    ifthen((\+var(X1)),(substring(X1,0,2,Part))),
    Part == $ات$,
    Out = $جمع مؤنث سالم مرفوع بالضمه الظاهره على اخرها$,
    rite([Out,' : ',X0,' : ',X1]),nl.
```

In the synthesis mode the variable Part will be assigned a value by the statement Part ==$ات$, which is considered as an output.

3.4 SYNTAX

To embed the Syntactic synthesis into the syntactic analyzer, we had to modify the eight sub-modules of the Syntactic Analyzer Module:

- Non-Article Nominal Simple
- Non-Article Verbal Simple
- Article Nominal Simple
- Article Verbal Simple
- Non-Article Nominal Compund
- Non-Article Verbal Compound
- Article Nominal Compund
- Article Verbal Compund.

3.4.1 The Syntactic Analyzer

The submodules have been augmented such that they work for both analysis and synthesis modes. There were three common changes that have been made for all the modules. These changes are:

1- Binding the variables (root, derivation, jns, hal, prefix, infix, suffex, and type) in all the predicates before using them in the synthesis mode. For example, in the following predicate

```
esm_eshara(L,NInfo,[Jns,Hal,$معرف$):-
  length(L,N),
  art_test(L,[Wrd,Rt,De,$اشاره$ اسم $,$$,Sfx],N)!],
  hal_jns(De,Jns,Hal),
  Ninfo=[Wrd,Rt,De,$اشاره$اسم$,Jns,Hal,$معرف$,$$,$$,$$,Sfx].
```

The L and the attributes Rt, De, Sfx are unbound during the synthesis mode. To bind the variables Rt, De, and Sfx, replace the variable Ninfo by its value in the first step. So after performing the replacement, the predicate is as shown below:

```
esm_eshara(L,[Wrd,Rt,De,$اشاره اسم$,Jns,Hal, $معرفه $,$$,$$,Sfx],
        [Jns,Hal,$معرفه $):-
        length(L,N),
        art_test(L,[Wrd,Rt,De,$اشاره اسم$,$$,Sfx],N)!],
        hal_jns(De,Jns,Hal).
```

2- The predicate ***length(L,N)*** must be avoided during the synthesis mode, however, at the same time this predicate is called during analysis mode. To resolve this problem we used the if-then statement to distinguish between the analysis and synthesis mode. During the analysis mode, the predicate ***length(L,N)*** has the variabl L bound, however, during the synthesis mode both variables L, and N are unbound variables. For example, the following predicate

```
esm_eshara(L,[Wrd,Rt,De,$اشاره اسم$,Jns,Hal,$معرفه $,$$,$$,Sfx],
        [Jns,Hal,$معرفه $):-
        length(L,N),
        art_test(L,[Wrd,Rt,De,$اشاره اسم$,$$,Sfx],N)!],
        hal_jns(De,Jns,Hal).
```

is modified to:

```
esm_eshara(L,[Wrd,Rt,De,$اشاره اسم$,Jns,Hal,$معرفه $,$$,$$,Sfx],
```

```
[Jns,Hal,$معرف$):-
ifthen( (\+ var(L)),(length(L,N)) ),
art_test(L,[Wrd,Rt,De,$اسم اشاره$,$$,Sfx],N)!],
hal_jns(De,Jns,Hal).
```

In the modified predicate the if-then statement distinguishes between the analysis and synthesis mode by testing the existence of the word L.

3- The interaction between the syntactic analyzer and the end-case analyzer takes place by inserting end-case calls in the syntactic analyzer before making any call to the morphological module. The function of these calls in the analysis mode is to find the end-case characteristics for each construct of a given sentence after making the calls to the morphological analyzer. However, in synthesis mode end-case characteristics for each construct are required before making any call to the morphological module, in order to construct each word. For this reason, an end-case call for each construct is inserted in the syntactic synthesizer before making the call to the morphological synthesizer. In inserting these calls, an if-then statement is used to prevent these calls during the analysis mode, as shown in the following example. For example, the following predicate

```
bfb_feil_2([L1|L2],[Nom1,Nom2,Nom3],Rem):-
[!feil(L1,[Type,_],Nom1,Lsfx1)!],
fael(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
tbi_fli(L3,Nom3,Jhmn3,Rem),
[X,Nom]=Nom2, asma_marfo(X,Nom).
```

is modified to:

```
bfb_feil_2([L1|L2],[Nom1,Nom2,Nom3],Rem):-
[!feil(L1,[Type,_],Nom1,Lsfx1)!],
ifthen((var(L2)),(
   [X,Nom]=Nom2,asma_marfo(X,Nom),Nom2=[X,Nom])),
fael(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
tbi_fli(L3,Nom3,Jhmn3,Rem),
[X,Nom]=Nom2, asma_marfo(X,Nom).
```

During the analysis mode the program will skip the first end-case calls by checking the condition ***var(L2)*** which will be false during this stage. However, in synthesis mode the condition ***var(L2)*** will be true and the end-case calls will be made.

3.5 Examples

EXAMPLE 1

The input to the preprocessor module is :

verb(feil([A,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,B,$$,C]),
fael([$اسم ال$,[D,$سلم$,$مفعل$,$سم$,$مذكر$,$جمع$,$معرفة$,E,$$,F]]),
mfol([$اسم ال$,[G,$كفر$,$فعال$,$سم$,$$,$$,$معرفة$,H,$$,I]])),

1- The preprocessor changes the structure of the given input into another structure as shown below, and it determines the following information: type of the sentence "فعلية بسيطة", type of the first construct "فعل", type of the second construct "فاعل", and the type of the last construct "مفعول به". The preprocessor calls the syntactic module by the following predicate:

bfb(L,[$فعلية بسيطة$,[$فعل$[A,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,B,$$,C]],
[$فاعل$ [$اسم ال$,[D,$سلم$,$مفعل$,$سم$,$مذكر$,$جمع$,$معرفة$,E,$$,F]]],
[$مفعول به$[$اسم ال$[G,$كفر$,$فعال$,$سم$,$$,$$,$معرفة$,H,$$,I]]],[])).

2- The syntactic module calls the end-case module with the following predicate:
feil_am($فعل$,[A,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,B,$$,C],_).

The end-case module determines the prefix "ي" of the word, and it calls the syntactic module by the following predicate with the new parameters:
feil_am($فعل$,[A,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$,],_).

3- The syntactic module calls the morphological module with the following predicate:
verb_test(L,[A,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$C] ,N).

The morphological module constructs the word "يقاتل", find the length of the word, and it calls the syntactic module with the following predicate:
verb_test([$يقاتل$],[$يقاتل$,$فتل$,$فاعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$,],5).

4- The syntactic module calls the end-case module with the following predicate:
asma_marfo($فاعل$,[$اسم ال$,[D,$سلم$,$مفعل$,$اسم$,$مذكر$,$جمع$,$معرفة$,E,$$,F]]).

The end-case module determines the suffix "ون", prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:
asma_marfo($فاعل$,[$اسم ال$,[D,$سلم$,$مفعل$,$اسم$,$مذكر$,$جمع$,$معرفة$,$ال$,$$,$ون$]]).

5- The syntactic module calls the morphological module with the following predicate:
noun_test([],[D,$سلم$,$مفعل$,$اسم$,$مذكر$,$جمع$,$معرفة$,$ال$,$$,$ون$],N).

The morphological module constructs the word "المسلمون" , find the length of the word, and it calls the syntactic module with the following predicate:
noun_test([$المسلمون$],

[$المسلمون$,$سلم$,$مفعل$,$اسم$,$مذكر$,$جمع$,$معرفة$,$ال$,$$,$ون$],8).

6- The syntactic module calls the end-case module with the following predicate:
asma_mnsb($مفعول به$[$اسم ال$],[G,$كفر$,$فعال$,$اسم$,$$,$$,$معرفة$,H,$$l]]).

The end-case module determines the prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:
asma_mnsb($مفعول به$[$اسم ال$],[G,$كفر$,$فعال$,$اسم$,$$,$$,$معرفة$,$ال$,$$,]]).

7- The syntactic module calls the morphological module with the following predicate:

noun_test([],[G,$كفر$,$فعال$,$اسم$,$$,$$,$معرفة$,$ال$,$$,$$]],N).

The morphological module constructs the word "الكفار", finds the length of the word, and it calls the syntactic module with the following predicate:

noun_test([$الكفار$],[$الكفار$,$كفر$,$فعال$,$اسم$,$$,$$,$معرفة$,$ال$,$$,$$]],6).

8- The output of the syntactic module is:

bfb($الكفار المسلمون يقاتل$],$فعلية بسيطة$,

[$فعل$],$يقاتل$],$قتل$],$فاعل$],$مضارع$],$مفرد$],$مذكر$],$غائب$],$ي$],$$],$$],

[$فاعل$], [$اسم ال$],$المسلمون$],$سلم$],$مفعل$],$اسم$],$مذكر$],$جمع$],$معرفة$],$ال$],$$],$ون$]]],

[$مفعول به$],[$اسم ال$],$الكفار$],$كفر$],$فعال$],$اسم$],$$],$$],$معرفة$],$ال$],$$],$$]],[]])

9- The syntactic module produces the sentence "يقاتل المسلمون الكفار".

EXAMPLE 2

The input to the preprocessor module is:

ma_t(verb(feil([A,$جعل$,$فعل$,$ماضي$,$مفرد$,B,$غائب$,C,,D]),
mfol([$اسم$,[E,$ورد$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,F,,G]])))

1- The preprocessor changes the structure of the given input into another structure as shown below, and it determines the following information: type of the subsentence "جملة تعجب", type of the first construct "فعل", type of the second construct "مفعول به". The preprocessor calls the syntactic module by the following predicate:

afb(L,[X,[$جملة تعجب$,[[H,2,$ما$,$اسم موصول$,I,J],
[$فعل$,[A,$جعل$,$فعل$,$ماضي$,$مفرد$,B,$غائب$,C,,D]],
[$مفعول به$,[$اسم الـ$,[E,$ورد$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,F,,G]]]]]],Rem).

2- The syntactic module calls the end-case module with the following predicate:
esm_marfo($اسم مبتدأ$,[H,2,$ما$,$اسم موصول$,I,J],_).

The end-case module calls the syntactic module by the following predicate with the new parameters:
esm_marfo($اسم مبتدأ$,[H,2,$ما$,$اسم موصول$,,],_).

3- The syntactic module calls the morphological module with the following predicate:
art_test(X,[H,2,$ما$,$اسم موصول$,,],N).

The morphological module constructs the word "ما", finds the length of the word "2", and it calls the syntactic module with the following predicate:
art_test([$ما$],[$ما$,2,$ما$,$اسم موصول$,,],2).

4- The syntactic module calls the end-case module with the following predicate:
feil_am($فعل$,[A,$جعل$,$فعل$,$ماضي$,$مفرد$,B,$غائب$,C,,D]).

The end-case module determines the prefix "ا" of the word, and it calls the syntactic module by the following predicate with the new parameters:

feil_am($فعل$,[A,$جمل$,$فعل$,$ماضي$,$مفرد$,B,$غائب$,$ا$,,]).

5- The syntactic module calls the morphological module with the following predicate:

verb_test(L,[A,$جمل$,$فعل$,$ماضي$,$مفرد$,_,$غائب$,$ا$,,],N).

The morphological module constructs the word "اجمل", finds the length of the word "4", and it calls the syntactic module with the following predicate:

verb_test([$اجمل$],[$اجمل$,$جمل$,$فعل$,$ماضي$,$مفرد$,_,$غائب$,$ا$,,],4)

6- The syntactic module calls the end-case module with the following predicate:

asma_mnsb($مفعول به$,[$اسم ال$,[E,$ورد$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,F,,G]]).

The end-case module determines the prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:

asma_mnsb($مفعول به$,[$اسم ال$,[E,$ورد$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]]).

7- The syntactic module calls the morphological module with the following predicate:

noun_test(L,[E,$ورد$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,],N).

The morphological module constructs the word "الورد", finds the length of the word "5", and it calls the syntactic module with the following predicate:

noun_test([$الورد$],[$ورد$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,],5)

8- The outupt of the syntactic module is:

afb([$ما أجمل الورد$],[$١ جملة تعجب$,[$فعلية ادواتية بسيطة مصدرة$,

[$ما$,2,$ما$,$اسم موصول$,,],

[$فعل$,[$اجمل$,$جمل$,$فعل$,$ماضي$,$مفرد$,_,$غائب$,$ا$,,],

[$مفعول به$,[$اسم ال$,[$ورد$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]]]]],[]).

9- The syntactic module produces the sentence "ماأجمل الورد".

EXAMPLE 3

The input to the preprocessor module is:

inna_n(nom(mbtd([$$,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,B,,C]),
khbr([$$,[D,$كرم$,$فعيل$,$اسم$,$$,$مفرد$,$نكرة$,E,$ي$,F])))

1- The preprocessor changes the structure of the given input into another structure as shown below, and it determines the following information: type of the sentence "ادواتية اسمية بسيطة", type of the first construct "حرف ناسخ", type of the second construct "مبتدأ", and the type of the last construct "خبر". The preprocessor calls the syntactic module by the following predicate:

bfb(L,[$ادواتية اسمية بسيطة$,[[A,2,$ان$,$حرف ناسخ$,,,],
[$مبتدأ$,[$اسم ال$,[B,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,C,,D]]],
[$خبر$,[$اسم نكرة$,[E,$كرم$,$فعيل$,$اسم$,$$,$مفرد$,$نكرة$,F,$ي$,G]]]]).

2- The syntactic module calls the morphological module with the following predicate:

art_test(L,[A,2,$ان$,$حرف ناسخ$,,,],N).

The morphological module constructs the word "ان", find the length of the word "2", and it calls the syntactic module with the following predicate:

art_test([$ان$],[$ان$,2,$ان$,$حرف ناسخ$,,,],2).

3- The syntactic module calls the end-case module with the following predicate:

asma_mansb($اسم ان$,[$اسم ال$,[B,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,C,,D]]).

The end-case module determines the prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:

asma_mansb($اسم ان$,[$اسم ال$,[B,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]]).

4- The syntactic module calls the morphological module with the following predicate:

noun_test(X,[$اسم ال$,[B,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]],N).

The morphological module constructs the word "الرجل", find the length of the word "5", and it calls the syntactic module with the following predicate:

noun_test([$الرجل$],

[$اسم ال$],[$ال$,$معرفة$,$مفرد$,$مذكر$,$اسم$,$فعل$,$رجل$,$ال+رجل$],,]],5).

5- The syntactic module calls the end-case module with the following predicate:

asma_marfo($خبر$,[$اسم نكرة$,[E,$كرم$,$فعل$,$اسم$,$$,$مفرد$,$تنكرة$,F,$ي$,G]]).

The end-case module calls the syntactic module by the following predicate with the new parameters:

asma_marfo($خبر$,[$اسم نكرة$,[E,$كرم$,$فعل$,$اسم$,$$,$مفرد$,$تنكرة$,,$ي$,]]).

6- The syntactic module calls the morphological module with the following predicate:

noun_test(X,[E,$كرم$,$فعل$,$اسم$,$$,$مفرد$,$تنكرة$,,$ي$,],N).

The morphological module constructs the word "كريم", find the length of the word "4", and it calls the syntactic module with the following predicate:

noun_test([$كريم$],[$كريم$,$كرم$,$فعل$,$اسم$,$$,$مفرد$,$تنكرة$,,$ي$,],4).

7- The output of the syntactic module is:

bfb(L,[$دواتية اسمية بسيطة$,[[$ان$,$ان$,2,$حرف ناسخ$,,,],

[$مبتدأ$,[$اسم ال$],[$ال+رجل$],$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]]],

[$خبر$,[$اسم نكرة$],[$كريم$,$كرم$,$فعل$,$اسم$,$$,$مفرد$,$تنكرة$,,$ي$,]]]]).

8- The syntactic module produces the sentence "ان الرجل كريم".

EXAMPLE 4

The input to the preprocessor is:

nom(mbtd([$ال$اسم$,[A,$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,B,,C]),
khbr(verb(feil([D,$دعو$,$فعل$,$ماضي$,$مفرد$,$مذكر$,$متكلم$,E,,F]),
fail([$ضمير$,[G,$ت$,$ت$,$ضمير متصل$,J,$مفرد$,$معرفة$,L,,M]),
mfol([$ضمر$,[N,$هـ$,$هـ$,$ضمير متصل$,$مذكر$,$مفرد$,$معرفة$,O,,P])))).

1- The preprocessor changes the structure of the given input into another structure as shown below, and it determines the following information: type of the first construct "مبتدأ", and the type of the second construct " جملة فعلية بسيطة". The preprocessor calls the syntactic module by the following predicate:

bfm(L,[X,[$مبتدأ$,[$اسم ال$],[A,$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,B,,C]]],
[$خبر$,[$فعلية بسيطة$],
[$فعل$,[D,$دعو$,$فعل$,$ماضي$,$مفرد$,$مذكر$,$متكلم$,E,,F]],
[$فاعل$,[$ضمير$,[G,$ت$,$ت$,$ضمير متصل$,J,$مفرد$,$معرفة$,L,,M]]],
[$مفعول به$,[$ضمر$,[N,$هـ$,$هـ$,$ضمير متصل$,$مذكر$,$مفرد$,$معرفة$,O,,P]]]]]).

2- The syntactic module calls the end-case module with the following predicate:
asma_marfo($مبتدأ$,[$اسم ال$],[A,$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,B,,C]).

The end-case module determines the prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:
asma_marfo($مبتدأ$,[$اسم ال$],[A,$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,J,,]]).

3- The syntactic module calls the morphological module with the following predicate:
noun_test(X,[A,$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,J,,],N).

The morphological module constructs the word "الرجل", find the length of the word "5", and it calls the syntactic module with the following predicate:
noun_test([$الرجل$],[$الرجل$],$رجل$,$فعل$,$سم$,$مذكر$,$مفرد$,$معرفة$,J,,],5).

4- The syntactic module calls the end-case module with the following predicate:
feil_am($فعل$,[D,$دعو$,$فعل$,$ماضى$,$مفرد$,$مذكر$,$متكلم$,E,,F]).

The end-case module determines the suffix "ته" of the word, and it calls the syntactic module by the following predicate with the new parameters:
feil_am($فعل$,[D,$دعو$,$فعل$,$ماضى$,$مفرد$,$مذكر$,$متكلم$,,,$ته$]).

5- The syntactic module calls the morphological module with the following predicate:
verb_test(X,[D,$دعو$,$فعل$,$ماضى$,$مفرد$,$مذكر$,$متكلم$,,,$ته$],N).

The morphological module constructs the word "دعوته", find the length of the word "5", and it calls the syntactic module with the following predicate:
verb_test(X,[$دعوته$,$دعو$,$فعل$,$ماضى$,$مفرد$,$مذكر$,$متكلم$,,,$ته$],5).

6- The syntactic module calls the end-case module with the following predicate:
asma_marfo($فاعل$,[$ضمير$,[G,$ت$,$ت$,$ضمير متصل$,J,$مفرد$,$معرفة$,L,,M]]).

The end-case module calls the syntactic module by the following predicate with the new parameters:
asma_marfo($فاعل$,[$ضمير$,[$ت$,$ت$,$ت$,$ضمير متصل$,J,$مفرد$,$معرفة$,,,]]).

7- The syntactic module calls the end-case module with the following predicate:
asma_mnsb($مفعول به$[$ضمير$,[N,$ه$,$ه$,$ضمير متصل$,$مذكر$,$مفرد$,$معرفة$,O,,P]]).

The end-case module calls the syntactic module by the following predicate with the new parameters:
asma_mnsb($مفعول به$[$ضمير$,[$ه$,$ه$,$ه$,$ضمير متصل$,$مذكر$,$مفرد$,$معرفة$,,,]]).

8- The output of the syntactic module is:

bfm([$الرجل دعوته$],[$اسمية مركبة$,

[$ال$,$معرفة$,$مفرد$,$مذكر$,$اسم$,$فعل$,$رجل$,$الرجل$],$اسم ال$],$مبتدأ$]

[$فعلية بسيطة$],$خبر$]

[$فعل$],[$دعوته$],$دعو$,$فعل$,$ماضي$,$مفرد$,$مذكر$,$متكلم$,,,$ته$]],

[$فاعل$],[$ضمير$],[$ت$,$ت$,$ت$,$ت$],$ضمير متصل$,ل,$مفرد$,$معرفة$,,,]],

[$مفعول به$],[$ضمر$],[$ه$,$ه$,$ه$,$ه$],$ضمير متصل$,$مذكر$,$مفرد$,$معرفة$,,,]]]]]]).

9- The syntactic module produces the sentence "الرجل دعوته".

EXAMPLE 5

The input to the preprocessor module is:

nom(mbtd([$اسم ال$,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,B,,C]]),
khbr(verb(feil([D,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,E,,F]),
fael([$ضمير$,[O,P,G,$ضمير مستتر$,H,I,$معرفة$,L,$$,K]]),
mfol([$اسم ال$,[L,$سار$,$فيعل$,$اسم$,$مؤنث$,$مفرد$,$معرفة$,M,$ي$,N])))))

1- The preprocessor changes the structure of the given input into another structure as shown below, and it determines the following information: type of the first construct "مبتدأ", and the type of the second construct " جملة فعلية بسيطة". The preprocessor calls the syntactic module by the following predicate:

bam(L,[X,[$مبتدأ$,[$اسم ال$,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,B,,C]]],
[$خبر$,[$فعلية بسيطة$,[$فعل$,[D,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,E,,F]],
[$فاعل$,[$ضمير$,[O,P,G,$ضمير مستتر$,H,I,$معرفة$,L,,K]]]
[$مفعول به$,[$اسم ال$,[L,$سار$,$فيعل$,$اسم$,$مؤنث$,$مفرد$,$معرفة$,M,$ي$,N]]],Rem)

2- The syntactic module calls the end-case module with the following predicate:
asma_marfo($مبتدأ$,[$اسم ال$,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,B,,C]]).

The end-case module determines the prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:
asma_marfo($مبتدأ$,[$اسم ال$,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,J,,]]).

3- The syntactic module calls the morphological module with the following predicate:
noun_test(X,[A,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,J,,],N).

The morphological module constructs the word "الرجل", find the length of the word "5", and it calls the syntactic module with the following predicate:
noun_test([$الرجل$],[$الرجل$,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,J,,],5).

4- The syntactic module calls the end-case module with the following predicate:
feil_am($فعل$,[D,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,E,$$,F],_).

The end-case module determines the prefix "ي" of the word, and it calls the syntactic module by the following predicate with the new parameters:
feil_am($فعل$,[D,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$],_).

5- The syntactic module calls the morphological module with the following predicate:
verb_test(X,[D,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$],N).

The morphological module constructs the word "يركب", find the length of the word "4", and it calls the syntactic module with the following predicate:
verb_test([$يركب$],[$يركب$,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$],4).

6- The syntactic module calls the end-case module with the following predicate:
asma_marfo($فاعل$,[$ضمير$,[O,P,G,$ضمير مستتر$,H,I,$معرفة$,L,$$,K]]).

The end-case module calls the syntactic module by the following predicate with the new parameters:
asma_marfo($فاعل$,[$ضمير$,[O,P,G,$ضمير مستتر$,H,I,$معرفة$,,$$,]]).

7- The syntactic module calls the end-case module with the following predicate:
asma_mnsb($مفعول به$,
[$اسم ال$,[L,$سار$,$فيعل$,$سم$,$مؤنث$,$مفرد$,$معرفة$,M,$ي$,N]]).

The end-case module determines the suffix "ة", prefix "ال" of the word, and it calls the syntactic module by the following predicate with the new parameters:
asma_mnsb($مفعول به$,
[$اسم ال$,[L,$سار$,$فيعل$,$سم$,$مؤنث$,$مفرد$,$معرفة$,$ال$,$ي$,$ة$]]).

8- The syntactic module calls the morphological module with the following predicate:
noun_test(X, [$اسم ال$,[L,$سار$,$فيعل$,$سم$,$مؤنث$,$مفرد$,$معرفة$,$ال$,$ي$,$ة$]],N).

The morphological module constructs the word "السيارة", find the length of the word "7", and it calls the syntactic module with the following predicate:

noun_test([$السيارة$],

[$اسم ال$],[$السيارة$],$سار$,$فعل$,$اسم$,$مؤنث$,$مفرد$,$معرفة$,$ال$,$ي$,$ة$]],7).

9- The output of the syntactic module is :

bam([$الرجل يركب السيارة$],[$اسمية مركبة$],

[$مبتدأ$,[$اسم ال$],$الرجل$,$رجل$,$فعل$,$اسم$,$مذكر$,$مفرد$,$معرفة$,$ال$,,]],

[$خبر$,[$فعلية بسيطة$],

[$فعل$,[$يركب$,$ركب$,$فعل$,$مضارع$,$مفرد$,$مذكر$,$غائب$,$ي$,$$,F]],

[$فاعل$,[$ضمير$,[O,P,G,$ضمير مستتر$,H,I,$معرفة$,,,]]]

[$مفعول به$,

[$اسم ال$],[$السيارة$],$سار$,$فعل$,$اسم$,$مؤنث$,$مفرد$,$معرفة$,$ال$,$ي$,$ة$]]],).

10- The syntactic module produces the sentence "الرجل يركب السيارة".

CHAPTER IV

CONCLUSION AND FUTURE WORK

In this research work we have designed and implemented a sentence generator (ASG) of Arabic sentences. The system can be easily geared towards any other related system or application since it has been built as a module.

The research consists of two components: design of ASG, and integration of ASG into NAUS.

ASG involves synthesis of morphological constructs, and syntax. As morphological constructs are affected by their syntactic roles and end-case grammar rules of Arabic, ASG incorporates end-case rules of Arabic in the construction of words and sentences.

ASG is composed of four main modules: preprocessor module, end-case module, morphological module, and syntactic module.

Integration of ASG into NAUS involves the determination of the interactions of ASG with other modules of NAUS. The syntactic

module in ASG, is the module that controls a proper sequence of interactions with other modules.

In the construction of a sentence, the syntactic module passes each element in the sentence along with its attribute list to the end-case module in order to determine its end-case mark. This end-case mark is added as an affix to the list of attributes of the element and is sent back to the corresponding syntactic module. Then the syntactic module forwards each element along with its new compiled attribute list to the morphological module to construct the word in its final shape.

With respect to the implementation of ASG, we took the advantage of the already developed modules of NAUS by integrating the generation modules (syntax, morphology, and end-case syntehesis) with corresponding analysis modules of NAUS.

The Arabic Sentence Generator was tested heavily using a large set of sentence meaning representations of different types and complexity. Section 3.5 shows some examples. The source code size of the ASG, including documentation, was about 541 KB.

For future work, it is suggested to integrate the generator into some other Arabic applications such as systems for teaching or learning Arabic, and translation systems. Furthermore, determination of the style of the text produced is an important area of research and requires further investigation.

APPENDIX (A)

Program Sample Segments

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*                  PREPROCESSOR MODULE PROGRAM                      */
/*                                                                   */
/*                                                                   */
/*-------------------------------------------------------------------*/


start :-
   knowledge,          /* get a sentence meaning representation */
   restructure_find_type(T2),
   syntactic_calls(Flag,T,T1).


restructure_find_type(L) :-
      key(I,K), nref(K,R), [! go_throught_the_list(R,[],L)!].


go_throught_the_list(R,C,T2) :-
   instance(R,T),
   ifthenelse((T==bfb;T==$خبر$;T==tajb;T==ann;T==afn;T==nan),
              (nab(C,T,T1)) , (add(C,T,T1))),!, nref(R,Nr)
      ifthen( (var(Flag)),(go_throught_the_list(Nr,T1,T2)) ).


nab(C,T,T1) :-
 ifthen((T=$خبر$;T=$فاعل$),([T,[D]]=[T,C], T2=[T,D], T1=[T2]) ),
 ifthen((T=bfb,[A,B,D]=C),(T2=[[$فعلية بسيطة$,[A,B,E]],D],T1=T2)),
 ifthen((T==bfb,var(T2)), (([A,B]=C,T2=[$فعلية بسيطة$,[A,B,_]],
      T1=[T2]);([A,B,D]=C,T2=[$فعلية بسيطة$,[A,B,D,_]],T1=[T2]))),
 ifthen((T==$اسمية بسيطة$),(([A,B,D,E]=C,T2=[$اسمية بسيطة$,A,B],
        T1=[T2,D,E]);(add(C,T,T2), T1=[T2]))),
 ifthen((T==nan), (add(C,N,T2), T1=[T2]) ),
 ifthen((T==aan; T==afn; T=nfn; T==ann),  (T2=[N,C], T1=[T2])),
 ifthen((T==tajb),([A,[B,[E,F,_]]]=C,T2=[$جملة تعجب$,[A,E,F,_]],
        T1=T2)).


syntactic_calls(Flag,T,[T1]) :-
  Flag=t, (
  (T== $اسمية بسيطة$, jmlh1(L,T1) );
  (T== bfb,   jmlh2(L,T1) );
  (T== nfn, ( jmlh7(L,T1);jmlh6(L,T1);jmlh2(L,T1);jmlh3(L,T1)) );
  (T== nan, ( jmlh4(L,T1);([T2,T3,T4]=T1,jmlh4(L,[T2,[T3,T4]]));
              jmlh8(L,T1);jmlh5(L,T1);jmlh1(L,T1)));
  ((T== afn; T==ann), (jmlh3(L,T1);jmlh7(L,T1)));
  ((T== aan; T==ann), (jmlh4(L,T1);jmlh8(L,T1)))),
```

```
verb(X,Y) :-member(X,[feil(_),fael(_)]),
           member(Y,[feil(_),fael(_),mfol(_)]),
           recorda(1,bfb,A), X,Y.

verb(X,Y) :-
           recorda(1,nfn,A), X,Y.

verb(X,Y,Z) :-
           member(X,[feil(_),fael(_)]),member(Y,[feil(_),
           fael( _)]),member(Z,[feil(_),fael(_),mfol(_)]),
           recorda(1,bfb,A), X,Y,Z.

verb(X,Y,Z) :-
           recorda(1,nfn,A), X,Y,Z.

nom(X,Y)  :-
           member(X,[mbtd([_:_]),khbr([_:_])]),
           member(Y,[mbtd([_ :_]),khbr([_:_])]),
           recorda(1,$اسمية بسيطة$,A), X,Y.

nom(X,Y)  :-
           recorda(1,nan,A1), X,Y.

nom(X,Y,Z)  :-
           recorda(1,nan,A1), X,Y,Z.

feil(X)   :- X=[_:_], Y=[$فعل$,X], recorda(1,Y,B8).

fael(X)   :  X=[_:_], Y=[$فاعل$,X], recorda(1,Y,Bb1).

fael(X)   :- recorda(1,$[فاعل$,B6), X,
             recorda(1,$|$,B9).

mbtd(X)   :- X=[_:_], Y=[$مبتدأ$,X], recorda(1,Y,Bb2).

khbr(X)   :- X=[_:_], Y=[$خبر$,X], recorda(1,Y,B7).

khbr(X)   :- recorda(1,$خبر$,B6), X.

mfol(X)   :- Y=[$مفعول به$,X], recorda(1,Y,Bb3).

atf(X) :- recorda(1,X,A).

jr_mjror(X) :- recorda(1,[$جار ومجرور$,X],B10).

esm_nkra(X) :- recorda(1,[$اسم نكرة$,X],B1).
```

```
bab(L,[$اسمية بسيطة$,[$مبتدا$,Nom],[$خبر$,Pred]],Rem) :-
            mbtda(L,Nom,[Jns1,Hal1,_],Rem1),
            khbr(Rem1,Pred,[Jns2,Hal2,_],Rem),
            mtchjh([Jns1,Hal1],[Jns2,Hal2],[Jns,Hal]).

mbtda(L,Nom,Jhmn,Rem) :-        rmbtda(L,Nom,Jhmn,Rem).

rmbtda([L1:L2],Ninfo,Jhmn,L2) :-  esm_alm(L1,Ninfo,Jhmn).

rmbtda([L1:L2],Ninfo,Jhmn,L2) :- esm_al(L1,Ninfo,Jhmn).

rmbtda([L1:L2],Ninfo,Jhmn,L2) :- dameer(L1,Ninfo,Jhmn).

esm_nkra(L,[$اسم نكرة$,
          [Wrd,Rt,De,$اسم$,Jns,Hal,$نكرة$,Prfx,Infx,Sfx]],
          [Jns,Hal,$نكرة$]) :-
          ifthen( (\+ var(L)) , (length(L,N)) ),
          noun_test(L,[Wrd,Rt,De,Hal,Jns,Prfx,Infx,Sfx],N),
          \+ member(Prfx,[$و$,$ب$,$ل$,$ك$,$ف$,$ال$,$لل$,$وب$,
          $ول$,$وك$,$فب$,$فل$,$فك$,$وال$,$الم$,
          $وكال$,$وبال$,$بال$,$كال$,$فلل$,$فال$,$ولل$,
          $وبال$,$فكال$]),
          \+ member(Sfx,[$ي$,$ك$,$ه$,$ها$,$هما$,$نا$,$هم$,$هن$]).

esm_eshara(L,[$اسم اشارة$,
             [Wrd,Rt,De,$اسم اشارة$,Jns,Hal,$معرفة$,$$,$$,Sfx]],
             [Jns,Hal,$معرفة$]) :-
             [! ifthen(( \+ var(L)),(length(L,N)) ),
             art_test(L,[Wrd,Rt,De,$اسم اشارة$,$$,Sfx],N)!],
             hal_jns(De,Jns,Hal).

esm_al(L,[$اسم ال$,[Wrd,Rt,De,$اسم$,Jns,Hal,$معرفة$,Prfx,Infx,Su]],
             [Jns,Hal,$معرفة$]) :-
             ifthen(\+ (var(L) ),( length(L,N)) ),
             noun_test(L,[Wrd,Rt,De,Hal,Jns,Prfx,Infx,Su],N),
             member(Prfx,[$ال$,$الم$,$المست$]).

esm_alm(L,[$اسم علم$,[Wrd,Wrd,Wrd,$اسم علم$,Jns,Hal,$معرفة$,$$,$$,$$]],
             [Jns,Hal,$معرفة$]) :-
             [!lst_word(L,Wrd),alm(Wrd,[Jns,Hal])!].

dameer(L,[$ضمير منفصل$,
            [Wrd,Rt,De,$ضمير منفصل$,Jns,Hal,$معرفة$,$$,$$,Sfx]],
            [Jns,Hal,$معرفة$]) :-
            [! ifthen( (\+ var(L)),(length(L,N)) ),
            art_test(L,[Wrd,Rt,De,$ضمير منفصل$,$$,Sfx],N),
            hal_jns(De,Jns,Hal)!].
```

```
bfb([L1:L2],[Nom1,Nom2,Nom3],Rem) :-
            [!feil(L1,[Typ,_],Nom1,Lsfx1)!],
            ifthen( ( var(L2)),
            ([X,Nom]=Nom2,asma_marfo(X,Nom),Nom2=[X,Nom] )),
            fael(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
            tbi_fli(L3,Nom3,Jhmn3,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom).

bfb([L1:L2],[Nom1,Nom2,Nom3,Nom4],Rem) :-
            [!feil(L1,[Typ,$متعدي1$],Nom1,Lsfx1)!],
            [! ifthen( ( var(L2)),
            ([X,Nom]=Nom2,asma_marfo(X,Nom),
            Nom2=[X,Nom] )) !],
            fael(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            [! ifthen( ( var(L2)),
            ([M,Nom7]=Nom3,asma_marfo(M,Nom7),
            Nom3=[M,Nom7] )) !],
            mfol(Lsfx2,L3,Nom3,Jhmn3,L4,[$$]),
            tbi_fli(L4,Nom4,Jhmn4,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7).

bfb_feil([L1:L2],[Nom1,Nom2,Nom3,Nom4,Nom5],Rem) :-
            [!feil(L1,[Typ,$متعدي2$],Nom1,Lsfx1)!],
            fael(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            mfol(Lsfx2,L3,Nom3,Jhmn3,L4,Lsfx3),
            mfol(Lsfx3,L4,Nom4,Jhmn4,L5,[$$]),
            tbi_fli(L5,Nom5,Jhmn5,Rem),
            [X,Nom]=Nom2,
            asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7),
            [M1,Nom77]=Nom4,
            asma_mnsb(M1,Nom77).

bfb_feil([L1:L2],[Nom1,Nom2,Nom3,Nom4,Nom5,Nom6],Rem) :-
            [!feil(L1,[Typ,$متعدي3$],Nom1,Lsfx1)!],
            fael(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            mfol(Lsfx2,L3,Nom3,Jhmn3,L3,Lsfx3),
            mfol(Lsfx3,L4,Nom4,Jhmn4,L5,Lsfx4),
            mfol(Lsfx4,L5,Nom5,Jhmn5,L6,[$$]),
            tbi_fli(L6,Nom6,Jhmn6,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M1,Nom7]=Nom3,
            asma_mnsb(M1,Nom7),
            [M2,Nom77]=Nom4,
            asma_mnsb(M2,Nom77),
            [M3,Nom777]=Nom5,
            asma_mnsb(M3,Nom777).
```

```
bfb_feil([L1|L2],[Nom1,Nom2,Nom3],Rem) :-
            [!feil(L1,[Typ,$متعدي1$],Nom1,Lsfx1)!],
            naeb_fael(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
            tbi_fli(L3,Nom3,Jhmn3,Rem),
            [X,Nom]=Nom2,
            asma_marfo(X,Nom).

bfb_feil([L1|L2],[Nom1,Nom2,Nom3,Nom4],Rem) :-
            [!feil(L1,[Typ,$متعدي2$],Nom1,Lsfx1)!],
            naeb_fael(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            mfol(Lsfx2,L3,Nom3,Jhmn3,L4,[$$]),
            tbi_fli(L4,Nom4,Jhmn4,Rem),
            [X,Nom]=Nom2,
            asma_marfo(X,Nom),
            [M1,Nom7]=Nom3,
            asma_mnsb(M1,Nom7).

feil(L,[Typ,Lm],[$فعل$,Finfo],Lsfx) :-
            ifthen( (\+ var(L)),
            (length(L,N)) ),
            verb_test(L,Finfo,N),
            xtrct(10,Finfo,Sfx),
            word_lst(Sfx,Lsfx),
            xtrct(4,Finfo,Typ).

fael(Lsfx,L,[$فاعل$,Nom],Jhmn,Rem,Lsfx) :-
            [! khbr(L,Nom,Jhmn,Rem)!].

fael(Lsfx,L,[$فاعل$,[$ضمير$,
            [_,_,_,$ضميرمستتر$,_,_,$معرفة$,$$,$$,$$]]],
            [Jns,Hal,$معرفة$],L,Lsfx) :-
            lst_word(Lsfx,Sfx),
            member(Sfx,
             [$$,$نيـ$,$هـ$,$ها$,$ك$,$هما$,$كما$,$نا$,
              $هم$,$هن$,$ك م$,$كه$,$كها$,$كهما$,$كهم$,
              $كهن$,$نيه$,$نيها$,$كموه$,$كموها$,$كموهم$,
              $كموهن$,$كموهما$]).

fael(Lsfx,L,[$فاعل$,Nom],Jhmn,Rem,Lsfx) :-
            [!khbr(L,Nom,Jhmn,Rem)!].

mfol([$$],L,[$مفعول به$,Nom],Jhmn,Rem,_) :-
            khbr(L,Nom,Jhmn,Rem).
```

```
aab(L,[$الدواتية اسمية بسيطة$,[[$مبتدأ$,Nom],[$خبر$,Pred]]],Rem) :-
          mbtda(L,Nom,[Jns1,Hal1,_],Rem1),
          khbr_dwt2(Rem1,Pred,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hal1],[Jns2,Hal2],[Jns,Hal]).

aab(L,[$الدواتية اسمية بسيطة$,[[$مبتدأ$,Nom],[$خبر$,Pred]]],Rem) :-
          mbt_dwt2(L,Nom,[Jns1,Hal1,_],Rem1),
          khbr_dwt1(Rem1,Pred,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hal1],[Jns2,Hal2],[Jns,Hal]).

aab(L,[$الدواتية اسمية بسيطة$,[[$خبر$,Pred],[$مبتدأ$,Nom]]],Rem) :-
          khbr_dwt2(L,Pred,[Jns1,Hal1,_],Rem1),
          khbr_dwt1(Rem1,Nom,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hal1],[Jns2,Hal2],[Jns,Hal]).

aab(L,[$الدواتية اسمية بسيطة$,[[$خبر$,Pred],[$مبتدأ$,Nom]]],Rem) :-
          khbr_dwt2(L,Pred,[Jns1,Hal1,_],Rem1),
          mbt_dwt1(Rem1,Nom,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hal1],[Jns2,Hal2],[Jns,Hal]).

aab_mwkda([L1:L2],[Nom1,Nom2],Rem) :-
          [!inna(L1,Nom1,[$$])!], ifthen( (var(L2)),(
          [_,[Y1,Y2],[Z1,Z2]]=Nom2, asma_mnsb($ان واسم$,Y2),
          asma_marfo($ان وخبر$,Z2), Nom2=[_,[Y1,Y2],[Z1,Z2]])),
          bab(L2,Nom2,Rem), adat(Nom1,_),
          [_,[Y1,Y2],[Z1,Z2]]=Nom2,
          asma_mnsb($ان واسم$,Y2),asma_marfo($ان وخبر$,Z2).

mbt_dwt1(L,Nom,Jhmn,Rem) :- mbtda(L,Nom,Jhmn,Rem).
mbt_dwt1(L,Nom,Jhmn,Rem) :- mbt_dwt2(L,Nom,Jhmn,Rem).

jr_mjror([L1:L4],[$جار ومجرور$,[Nom1,Nom2]],Rem) :-
          [!hrf_jr(L1,Nom1,L3),
          ifthenelse((L3==[$$]),(L2=L4),(L2=[L3:L4]))!],
          ifthen( (var(L4)),(asma_mjror($اسم مجرور$,Nom2)) ),
          mbtda(L2,Nom2,Jhmn2,Rem), adat(Nom1,_),
          asma_mjror($اسم مجرور$,Nom2).

eghra_dwt(L,[Nom1,Nom2,Nom3],Rem) :- [!mbt_dwt1(L,Nom1,Jhmn,L2),
          hrf_waw(L2,Nom2,L3)!],
          ifthen( (var(L)),(asma_mnsb($معطوف$,Nom3)) ),
          mbt_dwt1(L3,Nom3,Jhmn,Rem),
          asma_mnsb($مفعول به من فعل محذوف تقديره الزم$,Nom1),
          adat(Nom2,_), asma_mnsb($معطوف$,Nom3).

aab_estfhm([L1:L2],[[$مبتدأ$,Nom1],[$خبر$,Nom2]],Rem) :-
          ifthen( (var(L2)),(esm_marfo($مبتدأ$,Nom1,_)) ),
          esm_estfhm($من$,L1,Nom1),
          ifthen( (var(L2)),(asma_marfo($خبر$,Nom2)) ),
          khbr_dwt1(L2,Nom2,Jhmn,Rem),
          esm_marfo($مبتدأ$,Nom1,_),asma_marfo($خبر$,Nom2).
```

```
afb([L1|L2],[$فعلية ادواتية بسيطة$,[Nom1,Nom2,Nom3]|,Rem) :-
            [!feil(L1,[Typ,_],Nom1,Lsfx1)!],
            ifthen((var(L2)),
            ([X,Nom]=Nom2, asma_marfo(X,Nom),Nom2=[X,Nom])),
            fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
            ifthenelse( (var(L3),var(Nom3)),(Rem=L3),
            (tbi_fli_adwt(L3,Nom3,Jhmn3,Rem))),
            [X,Nom]=Nom2,asma_marfo(X,Nom).

afb([L1|L2],[$٢ فعلية ادواتية بسيطة$,[Nom1,Nom2,Nom3,Nom4]|,Rem):-
            [!feil(L1,[Typ,$متعدي1$],Nom1,Lsfx1)!],
            ifthen( (var(L2)),
               ([X,Nom]=Nom2,asma_marfo(X,Nom),Nom2=[X,Nom],
               [M,Nom7]=Nom3,asma_mnsb(M,Nom7), Nom3=[M,Nom 7])),
            fael_adwt(Lsfx1,L2,Nom2,Jhmn,L3,Lsfx2),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,[$$]),
            tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7).


afb([L1|L2],[$٣ فعلية ادواتية بسيطة$,
            [Nom1,Nom2,Nom3,Nom4,Nom5|],Rem) :-
            [!feil(L1,[Typ,$متعدي2$],Nom1,Lsfx])!],
            fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,Lsfx3),
            mfol_adwt(Lsfx3,L4,Nom4,Jhmn4,L5,[$$]),
            tbi_fli_adwt(L5,Nom5,Jhmn5,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7),
            [M2,Nom77]=Nom4, asma_mnsb(M2,Nom77).

afb([L1|L2],[$٤ فعلية ادواتية بسيطة$,
            [Nom1,Nom2,Nom3,Nom4,Nom5,Nom6]|,Rem) :-
            [!feil(L1,[Typ,$متعدي3$],Nom1,Lsfx1)!],
            fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,Lsfx3),
            mfol_adwt(Lsfx3,L4,Nom4,Jhmn4,L5,Lsfx4),
            mfol_dwt(Lsfx4,L5,Nom5,Jhmn5,L6,[$$]),
            tbi_fli_adwt(L6,Nom6,Jhmn6,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7),
            [M2,Nom77]=Nom4, asma_mnsb(M2,Nom77),
            [M22,Nom777]=Nom5, asma_mnsb(M22,Nom777).

afb([L1|L2],[$٥ فعلية ادواتية بسيطة$,[Nom1,Nom2,Nom3]|,Rem) :-
            [!feil(L1,[Typ,$متعدي1$],Nom1,Lsfx1)!],
            naeb_fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
            tbi_fli_adwt(L3,Nom3,Jhmn3,Rem),
            [X,Nom]=Nom2,
            asma_marfo(X,Nom).
```

```
afb([L1:L2],[$٦ فعلية ادواتية بسيطة$,[Nom1,Nom2,Nom3,Nom4]],Rem):-
            [!feil(L1,[Typ,$متعدي2$],Nom1,Lsfx1)!],
            naeb_fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2 ),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,[$$]),
            tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3,asma_mnsb(M,Nom7).

afb([L1:L2],[$٧ فعلية ادواتية بسيطة$,
            [Nom1,Nom2,Nom3,Nom4,Nom5]],Rem) :-
            [!feil(L1,[Typ,$متعدي3$],Nom1,Lsfx1)!],
            naeb_fael_adwt(Lsfx1,L2,Nom2,Jhmn2,L3,Lsfx2 ),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,Lsfx3),
            mfol_adwt(Lsfx3,L4,Nom4,Jhmn4,L5,[$$]),
            tbi_fli_adwt(L5,Nom5,Jhmn5.Rem),
            [X,Nom]=Nom2, asma_marfo(X,Nom),
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7),
            [M2,Nom77]=Nom4, asma_mnsb(M2,Nom77).

afb(L,[$١ فعلية ادواتية بسيطة مصدرة$,[$جملة تعجب$,Nom]],Rem) :-
            ma_tfdeel(L,Nom,Rem).

afb(L,[$٣ فعلية ادواتية بسيطة مصدرة$,[$جملة انـ$,Nom]],Rem) :-
            afb2_an(L,Nom,Rem).

afb(L,[$٤ فعلية ادواتية بسيطة مصدرة$,[$جملة استفهام$,Nom]],Rem):-
            afb2_estfhm(L,Nom,Rem).

afb(L,[$٥ فعلية ادواتية بسيطة مصدرة$,[$جملة قسم$,Nom]],Rem) :-
            afb2_qasam(L,Nom,Rem).

ma_tfdeel([L1,L2:L3],[Nom1,Nom2,Nom3,Nom4],Rem) :-
            ifthen( (var(L2)),(esm_marfo($مبتدأ$,Nom1,_),
            feil_am($فعل تفضيل تعجب$,Nom,_),
            Nom2 = [N,Nom])), ma(L1,Nom1),
            feil(L2,[$ماضي$,$متعدي3$],Nom2,Lsfx2),
            mfol_adwt(Lsfx2,L3,Nom3,Jhmn3,L4,_),
            tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
            esm_marfo($مبتدأ$,Nom1,_),
            [N,Nom]=Nom2, feil_am($فعل تفضيل تعجب$,Nom,_),
            irab_jmlh($جملة الفعل الماضي $,$ في محل رفع خبر $).

ma_tfdeel([L1,L2,L3:L4],[Nom1,Nom2,Nom3,Nom4],Rem) :-
            ma(L1,Nom1), feil(L2,[$ماضي$,_],Nom2,_),
            an(L3,Nom3), afb1(L4,Nom4,Rem),
            esm_marfo($مبتدأ$,Nom1,_), [N,Nom]=Nom2,
            f_madi(N,Nom,_),
            irab_jmlh($جملة الفعل الماضي $,$ في محل رفع خبر $),
            adat(Nom3,_), [N4,Nom44]=Nom4, f_mansob(N4,Nom44,_),
            irab_jmlh($ في محل نصب مفعول به لفعل التفضيل $,
            $والمصدر المؤول من انـ والفعل والفاعل$).
```

```
afb2_an([L1|L2],[[$مبتدأ$,Nom1,Nom2],[$خبر$,Nom3]|],Rem) :-
            an(L1,Nom1),
            ifthen( (var(L2)),([_,[E1,E2|_]]=Nom2,
            [F1,F2]=E1, f_mansob(F1,F2,_), E1=[F1,F2],
            Nom2=[_,[E1,E2|_]], asma_marfo($خبر$,Nom3,_))),
            bfb(L2,Nom2,L3),khbr_dwtl(L3,Nom3,Jhmn3,Rem),
            adat(Nom1,_),
            [_,[E1,E2|_]]=Nom2,
            [F1,F2]=E1,
            f_mansob(F1,F2,_),
            irab_jmlh($ في محل رفع مبتدأ $,
             $والمصدر المؤول من أن والفعل والفاعل$).
            asma_marfo($خبر$,Nom3,_).

afb2_an([L1|L2],[[$مبتدأ$,Nom1,Nom2],[$خبر$,Nom3]|],Rem) :-
           an(L1,Nom1), afb1(L2,Nom2,L3),
           khbr_dwtl(L3,Nom3,Jhmn3,Rem), adat(Nom1,_),
           [_,[ [E1,E2]|_]]=Nom2, [F1,F2]=E1,
           f_mansob(F1,F2,_),
           irab_jmlh($ في محل رفع مبتدأ $,
           $والمصدر المؤول من أن والفعل والفاعل$).
           asma_marfo($خبر$,Nom3,_).

afb2_an([L1,L2|L3],[[$مبتدأ$,Nom1,Nom2,Nom3],[$خبر$,Nom4]],Rem) :-
          an(L1,Nom1), la(L2,Nom2),
          afb1(L3,Nom2,L4), [[$لـ$|_],_]=Nom2,
          khbr_dwtl(L4,Nom4,Jhmn3,Rem),
          adat(Nom1,_),
          adat(Nom2,_),
          [N2,Nom22]=Nom2,
          f_mansob(N2,Nom22,_),
          irab_jmlh($ في محل رفع مبتدأ $,
          $والمصدر المؤول من أن والفعل والفاعل$).
          asma_marfo($خبر$,Nom4,_).

afb2_estfhm([L1|L2],[Ninfo,Nom],Rem) :-
          hl(L1,Ninfo), adat(Ninfo,_),
          ((bfb(L2,Nom,Rem),[_,[F1,F2]|_]=Nom,feil_am(F1,F2,_) );
          (afb1(L2,Nom,Rem),[_,[F1,F2]|_]=Nom,feil_am(F1,F2,_));
          (afb2_feil(L2,Nom,Rem),[[$أن$|_],Nom2]=Nom,
          irab_jmlh($ في محل رفع مبتدأ $,
          $والمصدر المؤول من أن والفعل والفاعل$)) ).

afb2_estfhm([L1|L3],[Ninfo,Nom],Rem) :-
           hamza(L1,Ninfo,L2),
           adat(Ninfo,_),
           ((bfb(L2,Nom,Rem),[_,[F1,F2]|_]=Nom,feil_am(F1,F 2,_));
            (afb1(L2,Nom,Rem),[_,[F1,F2]|_]=Nom,feil_am(F1, F2,_));
            (afb2_feil(L2,Nom,Rem),[[$أن$|_],Nom2]=Nom,
           irab_jmlh($ في محل رفع مبتدأ $,
           $والمصدر المؤول من أن والفعل والفاعل$)) ).
```

```
afb2_estfhm([L1|L2],[Ninfo,Nom],Rem) :-
          esm_estfhm($منذ$,L1,Ninfo), esm_marfo($مبتدا$,Ninfo,_),
          ((bfb(L2,Nom,Rem),[X,[[F1,F2]|_]]=Nom,feil_am(F1,F2,_));
          (afb1(L2,Nom,Rem),[X,[[F1,F2]|_]]=Nom,feil_am(F1,F2,_));
          afb2_feil(L2,Nom,Rem),[X|_]=Nom),
          irab_jmlh($في محل رفع خبر$,X).

afb2_estfhm([L1|L2],[Ninfo,Nom],Rem) :-
          esm_estfhm($ما$,L1,Ninfo),
          asma_mnsb($مفعول به مقدم$,Ninfo),
          ((bfb(L2,Nom,Rem),[X,[[F1,F2]|_]]=Nom,feil_am(F1,F2,_));
          (afb1(L2,Nom,Rem),[X,[[F1,F2]|_]]=Nom,feil_am(F1,F2,_));
          afb2_feil(L2,Nom,Rem),[X|_]=Nom).

afb2_qasam(L,[Nom1,Nom2],Rem) :-
          [!qasam(L,Nom1,Jhmn1,L2)!], afb2_feil(L2,Nom2,Rem),
          irab_jmlh($لا محل لها من الاعراب$,
                                        $ جملة جواب القسم $).

fael_adwt(Lsfx,L,Nom,Jhmn,Rem,Rsfx) :-
          fael(Lsfx,L,Nom,Jhmn,Rem,Rsfx).

fael_adwt(Lsfx,L,[$فاعل$,Nom],Jhmn,Rem,Rsfx) :-
          ifthen( (var(L)),(asma_marfo($فاعل$,Nom))),
          khbr_dwt3(L,Nom,Jhmn,Rem), asma_marfo($فاعل$,Nom).

mfol_adwt(Lsfx,L,[$مفعول به$,Nom],Jhmn,Rem,Rsfx) :-
          khbr_dwt1(L,Nom,Jhmn,Rem).

tbi_fli_adwt(L,[Nom1,Nom2],Jhmn,Rem) :-
          ifthenelse( (var(Nom1),var(L)),
          (tbi_fli(L2,Nom2,Jhmn,Rem)), (jr_mjror(L,Nom1,L2),
          tbi_fli(L2,Nom2,Jhmn,Rem))).

tbi_fli_adwt(L,Nom,Jhmn,Rem) :-
          tbi_fli(L,Nom,Jhmn,Rem).

adat_feil(L,Ninfo,[]) :-
          ifthen( (\+ var(L)),(length(L,N))),
          art_test(L,[Adat,Rt,De,Typ,$$,Sfx],N),
          member(Adat,[$قد$,$لا$,$لم$,$لن$,$سوف$,$ان$]),
          Ninfo=[Adat,Rt,De,Typ,$$,$$,Sfx].

ma(L,Ninfo) :-
          ifthen( (\+ var(L)),(length(L,N))),
          art_test(L,Ninfo,N), xtrct(1,Ninfo,Adat),Adat==$ما$.

la(L,Ninfo) :-
          ifthen( (\+var(L)),(length(L,N))),
          art_test(L,Ninfo,N), xtrct(1,Ninfo,Adat),Adat==$لا$.
```

```
bam(L,[$اسمية مركبة$,[$مبتدأ$,Nom],[$خبر$,Pred]],Rem) :-
          mbtda_mrkb(L,Nom,[Jns],Hall,_],Rem1),
          khbr_mrkb(Rem1,Pred,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hall],[Jns2,Hal2],[Jns,Hal]).

bam(L,[$اسمية مركبة$,[$مبتدأ$,Nom],[$خبر$,Pred]],Rem) :-
          mbtda_mrkb(L,Nom,[Jns1,Hall,_],Rem1),
          khbr(Rem1,Pred,[Jns2,Hal2,_],Rem),
          mtchjh([Jns1,Hall],[Jns2,Hal2],[Jns,Hal]).

bam(L,[$اسمية مركبة$,[$مبتدأ$,Nom],[$خبر$,Pred]],Rem) :-
          mbtda(L,Nom,[Jns1,Hall,_],Rem1),
          khbr_mrkb(Rem1,Pred,[Jns2,Hal2,_],Rem).

mbtda_mrkb([L1:L2],[$اسم مركب$,[Nom1,Nom2]],Jhmn,Rem) :-
          mowsol_esm(L1,Nom1,Jhmn1), khbr_jml(L2,Nom2,Rem),
          xtrct(1,Nom2,X), ifthen((X==$فعلية بسيطة$),
          (xtrct(2,Nom2,[[Y,Z]:_]), feil_am(Y,Z,_)) ),
          irab_jmlh($ جملة صلة لا محل لها من الاعراب $,X).

mbtda_mrkb(L,[$اسم مركب$,[Nom1,[$تابع$,Nom2],Nom3]],Jhmn,Rem) :-
          mbtda(L,Nom1,Jhmn1,[L2:L3]),
          esm_mowsol(L2,Nom2,Jhmn2), khbr_jml(L3,Nom3,Rem),
          xtrct(1,Nom3,X), ifthen((X==$فعلية بسيطة$),
          (xtrct(2,Nom3,[[Y,Z]:_]), feil_am(Y,Z,_))),
          irab_jmlh($ جملة صلة لا محل لها من الاعراب $,X).

khbr_mrkb(L,Nom,Jhmn,Rem) :- khbr_jml(L,Nom,Rem).

khbr_mrkb(L,Nom,Jhmn,Rem) :- khbr(L,Nom,Jhmn,Rem).
khbr_mrkb(L,Nom,Jhmn,Rem) :- mbtda_mrkb(L,Nom,Jhmn,Rem).

khbr_mrkb(L,[Nom1,Nom2],Jhmn,Rem) :-esm_nkra_mkrr(L,Nom1,Jhmn1,L1),
          khbr_jml(L1,[$صفة$,Nom2],Jhmn,Rem).

khbr_jml(L,Nom,Rem) :- bfb(L,Nom,Rem).

khbr_jml(L,Nom,Rem) :-
          ifthen( (var(L)),([_,[Y1,Y2],[Z1,Z2]]=Nom,
          asma_marfo(Y1,Y2),asma_marfo(Z1,Z2),
          [_,[Y1,Y2],[Z1,Z2]])), bab(L,Nom,Rem),
          [_,[Y1,Y2],[Z1,Z2]]=Nom,
          asma_marfo(Y1,Y2),asma_marfo(Z1,Z2).

mowsol_esm(L,[$اسم موصول$,Nom],[Jns,Hal,$معرفة$]) :-
        [! ifthen( (\+var(L)),(length(L,N))),art_test(L,Nom,N),
          xtrct(1,Nom,Wrd),
          member(Wrd,[$الذي$,$التي$,$اللذان$,$الذين$,$اللتان$,
                $اللتين$,$الذين$,$اللاتي$,$$,$اللائي$,
                $منْ$,$ما$]), bal_jns(Wrd,Jns,Hal)!].
```

```
bfm(L,[$1جملة فعلية مركبة$,[Nom1,Nom2]],Rem) :-
        ifthen( (var(L)),( xtrct(2,Nom1,[[_,Famr]:_]),
        [_,_,_,$أمر$:_]=Famr,
        xtrct(2,Nom2,[[_,[_,_,_,_,$مضارع$:_]]:_]),
        xtrct(2,Nom2,[[_,Z]:_]), xtrct(1,Nom2,X),
        feil_am($ فعل الطلب ، فعل الطلب $,Famr,_),
        feil_am($ فعل المقدرة ، جواب الطلب مجزوم بان $,Z,_))),
        xtrct(2,Nom1,[[_,Famr]:_]),
        [_,_,_,$أمر$:_]=Famr, bfb(L,Nom1,L1),
        xtrct(2,Nom2,[[_,[_,_,_,_,$مضارع$:_]]:_]),
        bfb(L1,Nom2,Rem),xtrct(2,Nom2,[[_,Z]:_]),
        xtrct(1,Nom2,X), feil_am($ فعل الطلب ، فعل الطلب $,Famr,_),
        feil_am($ فعل المقدرة ، جواب الطلب مجزوم بان $,Z,_),
        irab_jmlh($ جملة جواب الطلب لا محل لها من الاعراب $,X).

bfm([L1:L2],[$1جملة فعلية مركبة$,[Nom1,Nom2]],Rem) :-
        feil(L1,_,Nom1,Lsfx),
        ifthenelse((Lsfx==[$$]),(L3=L2),(L3 =[Lsfx:L2])),
        bam(L3,Nom2,Rem),xtrct(2,Nom2,[_,Fael]),
        asma_marfo($فاعل$,Fael),xtrct(3,Nom2,[_,Hal]),
        asma_mnsb($حال$,Hal),
        [Jml_Typ,[[_,[_,_,_,Typ:_]]:_]]=Hal,
        ifthen((Jml_Typ==$جملة فعلية بسيطة$),(Typ==$مضارع$)).

bfm([L1:L2],[$٢جملة فعلية مركبة$,[Nom1,Nom2]],Rem) :
        ifthen( (var(L1)),(
        [_,[_,Y],[_,Z]]=Nom2,asma_marfo($اسم كان$,Y),
        xtrct(3,Nom2,[_,Z]), asma_mnsb($خبر كان$,Z),
        Nom2=[_,[_,Y],[_,Z]] )), bam(L2,Nom2,Rem),
        [_,[_,Y],[_,Z]]=Nom2, asma_marfo($اسم كان$,Y),
        xtrct(3,Nom2,[_,Z]), asma_mnsb($خبر كان$,Z).

bfm([L1:L2],[$٢جملة فعلية مركبة$,[Nom1,[$جملة اسمية$,[$مبتدأ$,Nom2],
        [$خبر$,Nom3]]]],Rem) :-
        kan(L1,Nom1,Jhl), xtrct(2,Nom],[_,_,_,_,_,_,Sfx]),
        ifthen((Sfx==$ـه$,length(L1,3)),
        (Nom2=[_,_,_,$ضمير متصل$:_])),
        esm_kan(Sfx,Nom2,[Jns,Hal,$معرفة$]),
        khbr_mrkb(L2,Nom3,[Jns2,Hal2,_],Rem),
        asma_marfo($اسم كان$,Nom2), asma_mnsb($خبر كان$,Nom3).

bfm([L1:L2],[$٢جملة فعلية مركبة$,[Nom1,[$فاعل$,Nom2],Pred]],Rem) :-
        nema_besa(L1,Nom1),
        (mbtda(L2,Nom2,Jhmn2,L3);mbtda_mrkb(L2,Nom2,Jhmn2,L3)),
        (mbtda(L3,Pred,Jhmn3,Rem);mbtda_mrkb(L3,Pred,Jhmn3,Rem)),
        [Y1,Y2]=Nom1,f_madi(Y1,Y2,_),xtrct(2,Nom2,N),
        asma_marfo($فاعل$,N),
        irab_jmlh($ في محل رفع خبر مقدم
          $ الجملة من الفعل والفاعل $),
        xtrct(2,Pred,Nom33),
        asma_marfo($مبتدأ مؤخر$,Nom33).
```

```
aam(L,[$اسمية مركبة ادواتية$,Nom1,Nom2,Nom3|,Rem) :-
              a(L,Nom1,[L2:L5]),
              hrf_atf(L2,Nom2,L4),
              ifthenelse((L4==[$$]),(L3=L5),(L3=[L4:L5])),
              al(L3,Nom3,Rem).

aam(L,[$اسمية مركبة ادواتية$,Nom1,Nom2,Nom3|,Rem) :-
              a(L,Nom1,[L2:L5]),
              hrf_atf(L2,Nom2,L4),
              ifthenelse((L4==[$$]),(L3=L5),(L3=[L4:L5])),
              aam3(L3,Nom3,Rem).

a(L,Nom,Rem) :- bab(L,Nom,Rem),
              ifthen( (var(L)),( [_,[Y1,Y2],[Z1,Z2]]=Nom,
              asma_marfo(Y1,Y2),asma_marfo(Z1,Z2),
              Nom=[_,[Y1,Y2],[Z1,Z2]])),
              bab(L,Nom,Rem),
              [_,[Y1,Y2],[Z1,Z2]]=Nom,
              asma_marfo(Y1,Y2),asma_marfo(Z1,Z2).

a(L,Nom,Rem) :-
              ifthen( (var(L)),([_,[Y1,Y2],[Z,[Z1,Z2]]]=Nom,
              asma_marfo(Y1,Y2),
              Nom=[_,[Y1,Y2],[Z,[Z1,Z2]]]),
              aam1(L,Nom,Rem),
              [_,[Y1,Y2],[Z,[Z1,Z2]]]=Nom,
              asma_marfo(Y1,Y2),
              irab_jmlh($الجملة في محل رفع خبر$,Z).

khbr_jml_dwt(L,Nom,Rem) :- afb(L,Nom,Rem),
              xtrct(1,Nom,X),
              jml_typ(X,Y),
              ifthen((Y==$فعلية$),
              (xtrct(2,Nom,[[Y1,Z]:_]),feil_am(Y1,Z,_))).

khbr_jml_dwt(L,Nom,Rem) :-
              ifthen( (var(L)),([_,[Y1,Y2],[Z1,Z2]]=Nom,
              asma_marfo(Y1,Y2),
              asma_marfo(Z1,Z2),
              Nom=[_,[Y1,Y2],[Z1,Z2]] )),
              bam(L,Nom,Rem),
              [_,[Y1,Y2],[Z1,Z2]]=Nom,
              asma_marfo(Y1,Y2),
              asma_marfo(Z1,Z2).

aam2(L,[$اسمية مركبة ادواتية$,[$جملة ندب$,Nom1,Nom2]],Rem) :-
              [!wa_alef(L,Nom1,L1)!],
              mndb_mrkb(L1,Nom2,Jhmn1,Rem),
              adat(Nom1,_).
```

```
afm(L,[$ا دواتية فعلية مركبة$,Nom1,Nom2,Nom3],Rem) :-
          f(L,Nom1,[L2:L5]), hrf_atf(L2,Nom2,L4),
          ifthenelse((L4==[$$]),(L3=L5),(L3=[L4:L5])),
          f1(L3,Nom3,Rem).

afm(L,[$ا دواتية فعلية مركبة$,Nom1,Nom2,Nom3],Rem) :-
          f(L,Nom1,[L2:L5]), hrf_atf(L2,Nom2,L4),
          ifthenelse((L4==[$$]),(L3=L5),(L3=[L4:L5])),
          afm6(L3,Nom3,Rem).

afm([L1:L2],[Nom1,Nom2,Nom3],Rem) :-
          adat_jzm(Adat,L1,Nom1), feil_shrt(L2,Nom2,L3),
          jwb_shrt(L3,Nom3,Rem), adat(Nom1,_).

afm([L1:L2],[Nom1,Nom2,Nom3],Rem) :-
          adat_la_jzm(Adat,L1,Nom1), feil_shrt(L2,Nom2,L3),
          jwb_shrt(L3,Nom3,Rem), adat(Nom1,_).

afm(L,[Nom1,Nom2,Nom3],Rem) :-
          jwb_shrt(L,Nom1,[L1:L2]),
          adat_jzm(Adat,L1,Nom2),feil_shrt(L2,Nom3,Rem).

afm[L1:L2],[Nom1,Nom2,Nom3],Rem) :-
         jwb_shrt(L,Nom1,[L1:L2]),
         adat_la_jzm(Adat,L1,Nom2),feil_shrt(L2,Nom3,Rem).

afm(L,Nom,Rem) :- afm_estfhm(L,Nom,Rem).

f(L,Nom,Rem) :-
          bfb(L,Nom,Rem) ; afb(L,Nom,Rem); bfm(L,Nom,Rem);
          afm(L,Nom,Rem).

f1(L3,Nom3,Rem) :- f(L,Nom,Rem);a(L,Nom,Rem).

adat_jzm(Adat,L,Nom) :-
          ifthen((\+var(L)),(length(L,N))),
          xtrct(4,Nom,$اسم شرط$),
          art_test(L,Nom,N),xtrct(1,Nom,Adat),
          member(Adat,[$من$,$مهما$,$متى$,$أيا$,$أين$,
                  $أينما$,$كيف$,$أنى$,$حيثما$,$كيفما$,$أي$]).

afm_estfhm([L1:L3],[Ninfo,Nom],Rem) :-
          esm_estfhm($أ$,L1,Ninfo), adat(Ninfo,_), f_est(L2,Nom,Rem).

afm_estfhm([L1:L2],[Ninfo,Nom],Rem) :-
          esm_estfhm($متى$,L1,Ninfo), esm_mnsb($مفعول فيه$,Ninfo,_),
          f_est(L2,Nom,Rem).

afm_estfhm([L1:L2],[Ninfo,Nom],Rem) :-
          esm_estfhm($أيان$,L1,Ninfo), esm_mnsb($مفعول فيه$,Ninfo,_),
          f_est(L2,Nom,Rem).
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*                    NOUN END-CASE SUB-MODULE                       */
/*                                                                   */
/*-------------------------------------------------------------------*/


% %%%%%%%%%% % الأسماء الخمسة
esm_marfo(X0,[X1,X2,_,$اسم$,_,$مفرد$,_,_,_,_,Part], Out ) :-
     member(X2,[$ذو$,$فو$,$حمو$,$أبو$,$أخو$]),
     ifthen( (\+ var(X1)) , (substring(X1,0,1,Part)) ),
     Part== $و$,
     Out = $اسم من الأسماء الخمسة مرفوع بالواو$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
% اسم اشارة
%%%%%%%%%%
esm_marfo(X0,[X1,_,X3,$اسم اشارة$,_,$مثنى$,_,_,_,_ ], Out ) :-
     X1 = X3,
     member(X1, [$هذان$,$هاتان$,$ذانك$,$تانك$,$ذانتك$,$تانتك$] ),
     Out = $اسم اشارة مبني على الألف لأنه مثنى في محل رفع$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,_,X3,$اسم اشارة$,_,_,_,_,_,_ ], Out ) :-
     member(X1, [$هذا$,$هنا$,$هكذا$]), X1 = X3,
     Out = $اسم اشارة مبني على السكون في محل رفع$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,_,X3,$اسم اشارة$,_,_,_,_,_,_ ], Out ) :-
     X1 = X3,
     member(X1,[$ذاك$,$ذلك$,$أولئك$,$أولالك$,$هناك$,
                $هنالك$,$ثم$,$ثمة$,$تلك$,$تيك$]),
     Out = $اسم اشارة مبني على الفتح في محل رفع$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
% اسم علم مفرد
%%%%%%%%%%%%%%%%%%%%
esm_marfo(X0,[X1,X2,_,$اسم علم$,_,$مفرد$,_,_,_,_], Out ) :-
     Out = $اسم علم مرفوع بضمة مقدرة للثقل$,
     ifthen( (\+var(X1)),(rite([Out,' : ',X0,' : ',X1] ), nl)),
     X1 = X2.

esm_marfo(X0,[X1,X2,_,$اسم علم$,_,$مفرد$,_,_,_,_], Out ) :-
     ifthenelse( (\+ var(X1)) , (substring(X1,0,1,Part),
     (Part == $ا$ ;
      Part == $ى$),
     Out = $اسم علم مرفوع بضمة مقدرة للتعذر$,
     rite([Out,' : ',X0,' : ',X1] ), nl ),
     (X1 == X2)).
```

```
%%%%%%%%%%%%%%%%%%%%
%اسم موصول
%%%%%%%%%%%%%%%%%%%%
esm_marfo(X0,[X1,_,X3,$اسم موصول$,_,_], Out ) :-
     member( X1, [$الذان$]),
     X1 = X3,
     Out = $اسم موصول مبني على الالف في محل رفع$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
% اسم العدد
%%%%%%%%%%%%%%%%%%%%
esm_marfo(X0,[X1,$عشرة$,_,$اسم عدد$,$مؤنث$,_,_,_,_,_ ], Out ) :-
     member( X1, [$عشر$]),
     Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$سبعون$,_,$اسم عدد$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$سبعون$]),
     Out = $ اسم عدد مرفوع بالواو لانه ملحق بجمع المذكر السالم$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$ثمانون$,_,$اسم عدد$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$ثمانون$]),
     Out = $ اسم عدد مرفوع بالواو لانه ملحق بجمع المذكر السالم$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$تسعون$,_,$اسم عدد$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$تسعون$]),
     Out = $ اسم عدد مرفوع بالواو لانه ملحق بجمع المذكر السالم$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$واحد$,_,$اسم عدد$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$واحد$,$احدى$]),
     Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$اربع$,_,$اسم عدد$,$مؤنث$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$اربع$]),
     Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$خمس$,_,$اسم عدد$,$مؤنث$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$خمس$]),
     Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,' : ',X0,' : ',X1] ), nl.

esm_marfo(X0,[X1,$خمس$,_,$اسم عدد$,$مذكر$,_,_,_,_,_,_ ], Out ) :-
     member( X1, [$خمسة$]),
     Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,' : ',X0,' : ',X1] ), nl.
```

```
esm_marfo(X0,[X1,$ست$,_,$اسم عدد$,$مؤنث$,_,_,_,_,_ |, Out ) :-
      member( X1, [$ست$]),
      Out = $اسم عدد مرفوع بالضمة الظاهرة على آخره$,
      rite([Out,' : ',X0,' : ',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
حالة المضاف تحتاج الى معالجة خاصة #  جمع المذكر السالم %
%%%%%%%%%%%%%%%%%%%%
esm_marfo(X0,[X1,_,X3,$اسم$,$مذكر$,$جمع$,_,_,_,Part|, Out ) :-
      ifthen( (\+ var(X1)) , (substring(X1,0,1,Part)) ),
      Part = $و$,
      Out = $جمع مذكر سالم مرفوع بالواو$,
      rite([Out,' : ',X0,' : ',X1] ), nl.

esm_mansob(X0,[X1,X2,$اسم$,_,$مفرد$,_,_,_,_ |, Out ) :-
      member(X2,[$ب$,$خ$,$ح$,$ف$,$ذ$]),
      member(X1,[$با$,$أخا$,$حما$,$فا$,$ذا$]),
      ifthen( (\+ var(X1)) , (substring(X1,1,1,Part),
      Part == $ا$ ,
      Out = $اسم من الأسماء الخمسة منصوب بالألف$,
      rite([Out,' : ',X0,' : ',X1] ), nl)).

%%%%%%%%%%%%%%%%%%%%
%   اسم علم ينتهي ب ا  ى   %
%%%%%%%%%%%%%%%%%%%%
      esm_mansob(X0,[X1,X2,_,$اسم علم$,_,$مفرد$,_,_,_,_], Out ) :-
      ifthenelse( (\+ var(X1)) , ( substring(X1,0,1,Part),
      (Part == $ا$ ;
       Part == $ى$),
      Out = $اسم علم منصوب بفتحة مقدرة للتعذر$,
      rite([Out,' : ',X0,' : ',X1] ), nl),
      (X1==X2)).

%%%%%%%%%%%%%%%%%%%%
% علم مركب مزجي %
%%%%%%%%%%%%%%%%%%%%
esm_mansob(X0,[X1,_,_,$اسم علم$,_,$مفرد$,_,_,_,_ |, Out ) :-
      member( X1, [$حضرموت$,$بيت لحم$,$بعلبك$]),
      Out = $اسم علم مركب مزجي منصوب بالفتحة الظاهرة على آخره$,
      rite([Out,' : ',X0,' : ',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
% المثنى %
%%%%%%%%%%%%%%%%%%%%
esm_mansob(X0,[X1,_,_,$اسم$,_,SDP,_,_,_,Part], Out ) :-
      member(SDP,[$ $,$مثنى$]),
      ifthen( (\+ var(X1)) , ( substring(X1,0,1,Part)) ),
      Part == $ي$ , Out = $ مثنى منصوب بالياء $,
      rite([Out,' : ',X0,' : ',X1] ), nl.
```

```
/%-------------------------------------------------------------------%/
/%                                                                   %/
/%                        VERB END-CASE SUBMODULE                    %/
/%                                                                   %/
/%-------------------------------------------------------------------%/
f_madi(X0,[X1,X2,X3,$ماضي$,_,_,Sfx,_,_,_], Out ) :-
     member(Sfx,[$$,$ه$,$ها$,$هما$,$هن$,$هم$,$نا$,$ني$,$ك$,$كما$,
             $كم$,$كن$]), string_length(Sfx, Lnth2 ),
     ifthen( (\+ var(X1)) , ( string_length(X1, Lnth1 ),
         Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X),
         substring(X,0,1,H))), (H==$ك$;H==$ا$),
     Out = $ ماضي مبني على الفتحة المقدرة للتعذر $,
     rite([Out,X0,':',X,':',X1] ), nl.

%%%%%%%%%%%%%%%%%%%%
%%   الضمير مفعول
%%%%%%%%%%%%%%%%%%%%
f_madi(X0,[X1,X2,X3,$ماضي$,_,_,Sfx,$مذكر$,$مفرد$,$غائب$], Out ) :-
    member(Sfx,[$$,$نا$,$ه$,$ها$,$هما$,$هن$,$هم$,$ني$,$ك$,$كما$,
                $كم$,$كن$]), string_length(Sfx, Lnth2 ),
    ifthen( (\+ var(X1)) , (
        string_length(X1, Lnth1 ),
        Lnth is Lnth1 - Lnth2,
        substring(X1,Lnth2,Lnth,X))),
    Out = $ ماضي مبني على الفتحه $.

f_madi(X0,[X1,X2,X3,$ماضي$,_,_,Sfx,Jns,_,_], Out ) :-
    ifthen( (\+ var(X1)) , ( string_length(Sfx, Lnth2 ),
    string_length(X1, Lnth1 ), Lnth is Lnth1 - Lnth2,
    substring(X1,Lnth2,Lnth,X))),
    Out = $ ماضي مبني على السكون لاتصاله بضمير رفع متحرك $,
    rite([Out,X0,':',X,':',X1] ), nl.

f_amr(X0,[X1,X2,X3,$امر$,_,_,Sfx,_,_,_], Out ) :-
    substring(X2,0,1,H),
    member(H,[$ي$,$ا$,$و$,$ك$]),
    ifthen( (\+ var(X1)) , ( string_length(Sfx, Lnth2),
        string_length(X1, Lnth1 ),
        Lnth is Lnth1 - Lnth2  ,
        substring(X1,Lnth2,Lnth ,X))),
    Out = $  امر مبني على حذف حرف العلة  $,
    rite([Out,X0,':',X,':',X1] ), nl.

f_amr(X0,[X1,X2,X3,$امر$,_,_,Sfx,_,_,_], Out ) :-
   member(Sfx,[$$,$ه$,$ها$,$هما$,$هن$,$هم$,$نا$,$ني$,$ك$,$كما$,$كم$
              ,$كن$]), string_length(Sfx, Lnth2),
      ifthen( (\+ var(X1)) , ( string_length(X1, Lnth1 ),
      Lnth is Lnth1 - Lnth2,
      substring(X1,Lnth2,Lnth,X))),
    Out = $ امر مبني على السكون $,
    rite([Out,X0,':',X,':',X1] ), nl.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% الأمر يبنى على السكون مع النون
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f_amr(X0,[X1,X2,X3,$أمر$,_,_,Sfx,_,_,_], Out ) :-
    ifthen( (\+ var(X1)) , (
      string_length(Sfx, Lnth2 ), Lnth2 >=1,
      B is Lnth2-1,
      substring(Sfx,B,1,H),
      H==$ن$, % اذا كانت اول حروف اللاحقة للفعل ن فهو للجمع المؤنث
       string_length(X1, Lnth1 ),
       Lnth is Lnth1 - Lnth2 ,
       substring(X1,Lnth2,Lnth ,X))),
     Out = $أمر مبني على السكون$,
     rite([Out,X0,':',X,':',X1] ), nl.

f_majzom(X0,[X1,X2,X3,$مضارع$,_,_,Sfx,Jns,Hal,_], Out ) :-
     ( member(Sfx,[$ا$,$وا$,$ي$]);
     (substring(Sfx,Lnth2-1,1,H),member(H,[$ا$,$ي$,$و$]) ) ),
     string_length(Sfx, Lnth2 ), Lnth2 >=1,
    ifthen( (\+ var(X1)) , ( string_length(X1, Lnth1 ),
     Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X))),
     Out = $مضارع مجزوم بحذف النون لأنه من الأفعال الخمسة$,
     rite([Out,X0,':',X,':',X1] ), nl.

f_majzom(X0,[X1,X2,X3,$مضارع$,_,_,Sfx,Jns,Hal,_], Out ) :-
    substring(X2,0,1,H),
    member(H,[$ا$,$ى$,$ي$,$و$]),
   ifthen( (\+ var(X1)) , (
    string_length(Sfx, Lnth2 ),string_length(X1, Lnth1 ),
    Lnth is Lnth1 - Lnth2,substring(X1,Lnth2,Lnth,X))),
    Out = $مضارع مجزوم بحذف حرف العلة$,
    rite([Out,X0,':',X,':',X1] ), nl.

f_mansob(X0,[X1,X2,X3,$مضارع$,_,_,Sfx,Jns,Hal,_], Out ) :-
    ( member(Sfx,[$ا$,$وا$,$ي$]);
    (substring(Sfx,Lnth2-1,1,H),member(H,[$ا$,$ي$,$و$]) ) ),
    string_length(Sfx, Lnth2 ), Lnth2 >=1,
    ifthen( (\+ var(X1)) , ( string_length(X1, Lnth1 ),
     Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X))),
     Out = $مضارع منصوب بحذف النون لأنه من الأفعال الخمسة$,
     rite([Out,X0,':',X,':',X1] ), nl.


f_mansob(X0,[X1,X2,X3,$مضارع$,_,_,Sfx,$مؤنث$, Hal,_], Out ) :-
    ifthen( (\+ var(X1)) , ( string_length(Sfx, Lnth2 ),
    Lnth2 >=1, B is Lnth2-1, substring(Sfx,B,1,H),
    H==$ن$, string_length(X1, Lnth1 ),
    Lnth is Lnth1 - Lnth2,
    substring(X1,Lnth2,Lnth,X))),
    Out = $مضارع مبني على السكون في محل نصب لاتصاله بنون النسوة$,
    rite([Out,X0,':',X,':',X1] ), nl.
```

```
f_marfo(X0,[X1,X2,X3,$مضارع$,$ا$,_,Sfx,_,$مفرد$,$متكلم$|,Out):
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ت$,_,Sfx,$مذكر$,$مفرد$,$مخاطب$|,Out):-
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ي$,_,Sfx,$مذكر$,$مفرد$,$غائب$|,Out):-
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ت$,_,Sfx,$مؤنث$,$مفرد$,$مخاطب$|,Out):-
     (Sfx = $ى$; Sfx=$$),
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ت$,_,Sfx,$مؤنث$,$مفرد$,$غائب$|,Out):-
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ن$,_,Sfx,_,X,$متكلم$|, Out):-
     (X=$مثنى$; X= $جمع$),
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ت$,_,Sfx,_,$مثنى$,$مخاطب$|, Out ) :-
     (Sfx = $ا$; Sfx=$ان$),
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ي$,_,Sfx,$مذكر$,$مثنى$,$غائب$|,Out) :-
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,$ت$,_,Sfx,$مؤنث$,$مثنى$,$غائب$|,Out) :-
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,Prfx,_,Sfx,$مذكر$,$جمع$,X|, Out ) :-
     ( (X=$مخاطب$,Prfx=$ت$,(Sfx=$ا$, Sfx=$ون$));
     (X= $غائب$,Prfx=$ي$,(Sfx=$ا$,Sfx=$ون$)) ),
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.

f_marfo(X0,[X1,X2,X3,$مضارع$,Prfx,_,$ن$,$مذكر$,$جمع$,X|, Out ) :-
     ( (X=$مخاطب$,Prfx=$ت$);
     (X= $غائب$,Prfx=$ي$) ),
     Out = $مضارع مرفوع بالضمة الظاهرة على آخره$,
     rite([Out,X0,':',X,':',X1| ), nl.
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*                    ARTICLE SUBMODULE                              */
/*                                                                   */
/*-------------------------------------------------------------------*/

art_test(L,T,2) :- art_test2(L,T).
art_test(L,T,3) :- art_test3(L,T).
art_test(L,T,4) :- art_test4(L,T).
art_test(L,T,5) :- art_test5(L,T).
art_test(L,T,6) :- art_test6(L,T).
art_test(L,T,7) :- art_test7(L,T).
art_test(L,T,8) :- art_test8(L,T).
art_test(L,T,9) :- art_test9(L,T).

 /*       ahQMdG          */
art_test2([A,B],[Oword,TC,Root,Type,PR,SU]) :-
           find_art2([A,B],TC,Root,Type,PR,SU),
           concat([A,B],Oword).

art_test3([A,B,C],T) :- member(C,[$h$,$a$]),
           find_art2([A,B],TC,Root,Type,PR,SU),
           concat([A,B,C],Oword),
           conca(PR,C,X),
           T = [Oword,TC,Root,Type,X,SU].

 art_test3([A,B,C],[Oword,TC,Root,Type,PR,SU]) :-
           find_art3([A,B,C],TC,Root,Type,PR,SU),
           concat([A,B,C],Oword).

 art_test4([A,B,C,D],[Oword,TC,Root,Type,PR,SU]) :-
           find_art4([A,B,C,D],TC,Root,Type,PR,SU),
           concat([A,B,C,D],Oword).

 art_test5([A,B,C,D,E],[Oword,TC,Root,Type,PR,SU]) :-
           member(E,[$h$,$a$]),
           find_art4([A,B,C,D],TC,Root,Type,PR,SU),
           concat([A,B,C,D,E],Oword),
           conca(PR,E,X).

 art_test6([A,B,C,D,E,F],[Oword,TC,Root,Type,PR,SU]) :-
           find_art6([A,B,C,D,E,F],TC,Root,Type,PR,SU),
           concat([A,B,C,D,E,F],Oword).


art_test8([A,B,C,D,E,F,G,H],[Oword,TC,Root,Type,X,SU]) :-
           member(H,[$h$,$a$]),
           ifthen( (var(A)), ( conca(PR,H,X)) ),
           find_art7([A,B,C,D,E,F,G],TC,Root,Type,PR,SU),
           concat([A,B,C,D,E,F,G,H],Oword),
           concat(PR,H,X).
```

```
art_test9([A,B,C,D,E,F,G,H,I],[Oword,TC,Root,Type,X,SU]) :-
          member(I,[$h$,$a$]),
          ifthen( (var(A)), (conca(PR,I,X))),
          find_art8([A,B,C,D,E,F,G,H],TC,Root,Type,PR,SU),
          concat([A,B,C,D,E,F,G,H,I],Oword),
          conca(PR,I,X).

find_art2([A,B],2,$hg$,$dUHe QjeV$,B,$$)  :-
          member(Root,[$gH$,$gd$]),
          conca([A,B],Root).

find_art2([A,B],2,Root,$dhUhe eSG$,$$,$$)  :-
          member(Root,[$jG$,$Ge$,$fe$!),
          conca([A,B],Root).

find_art2([A,B],2,Root,$jaf aQM$,$$,$$)  :-
          member(Root,[$Gd$,$Ge$]),
          conca([A,B],Root).

find_art2([A,B],2,Root,$QOUe aQM$,$$,$$)  :-
          member(Root,[$fG$,$Ge$,$je$,$fG$,$hd$]),
          conca([A,B],Root).

find_art3([A,B,C],2,Root,$IQGTG eSG$,C,$$)  :
          member(C,[$H$,$d$,$c$]),
          member(Root,[$GP$,$iJ$,$gJ$,$jP$,$IP$]),
          conca([A,B],Root).

find_art3([A,B,C],2,Root,$dUJe QjeV$,C,$$)  :-
          member(C,[$H$,$d$]),
          member(Root,[$Gg$,$eg$,$fg$!),
          conca([A,B],Root).

find_art3([A,B,C],2,Root,$AGfKJSG aQM$,$$,$$) :-
          Root=$GdG$,
          conca([A,B,C],Root).

find_art4([A,B,C,D],2,Root,$dhUhe eSG$,$$,$$)  :
          member(Root,[$jPdG$,$iJdG$,$GPGe$]),
          conca([A,B,C,D],Root).

find_art6([A,B,C,D,E,F],2,Root,$dhUhe eSG$,F,$$)  :-
          member(F,[$H$,$d$,$c$]),
          member(Root,[$fjPdG$,$jdCdG$]),
          conca([A,B,C,D,E],Root).
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                    NOUN SUBMODULE                           */
/*                                                             */
/*-------------------------------------------------------------*/
   noun_test(L,T,2)  :- noun_test2(L,T).
   noun_test(L,T,3)  :- noun_test3(L,T).
   noun_test(L,T,4)  :- noun_test4(L,T).
   noun_test(L,T,5)  :- noun_test5(L,T).
   noun_test(L,T,6)  :- noun_test6(L,T).
   noun_test(L,T,7)  :- noun_test7(L,T).
   noun_test(L,T,8)  :- noun_test8(L,T).
   noun_test(L,T,9)  :- noun_test9(L,T).
   noun_test(L,T,10) :- noun_test10(L,T).
   noun_test(L,T,11) :- noun_test11(L,T).
   noun_test(L,T,12) :- noun_test12(L,T).
   noun_test(L,T,13) :- noun_test13(L,T).

  noun_test2([A,B],[Oword,$فـل$,$مفرد$,$$,$$,$$,$$]):-
            conca([A,B],Oword).

  noun_test3([A,B,C],[Oword,Oword,$فعـل$,$مفرد$,$$,$$,$$,$$])  :-
            conca([A,B,C],Oword).

  noun_test3([A,B,C],[Oword,Oword,DE,$مفرد$,$$,C,$$,$$])  :-
            member(C,[$و$,$ـفـ$,$ـئـ$,$ـٰـ$,$ـبـ$,$ـلـ$,$ا$]),
            conca([A,B,C],Oword), concat([$فـل$,C],DE).

  noun_test4([A,B,C,D],[Oword,RO,DE,SDP,MF,PR,IN,SU])  :-
            npre_test4([A,B,C,D],RO,DE,SDP,MF,PR,IN,SU),
            concat([A,B,C,D],Oword).

  noun_test4([A,B,C,D],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
            nsuf_test4([A,B,C,D],RO,DE,SDP,MF,PR,IN,SU),
            concat([A,B,C,D],Oword).

  noun_test4([A,B,C,D],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
            ninf_test4([A,B,C,D],RO,DE,SDP,MF,PR,IN,SU),
            concat([A,B,C,D],Oword).

  noun_test5([A,B,C,D,E],[Oword,RO,DE,SDP,MF,PR,IN,SU])  :-
            nsuf_test5([A,B,C,D,E],RO,DE,SDP,MF,PR,IN,SU),
            concat([A,B,C,D,E],Oword).

  noun_test5([A,B,C,D,E],[Oword,RO,DE,SDP,MF,PR,IN,SU])  :-
            npre_test5([A,B,C,D,E],RO,DE,SDP,MF,PR,IN,SU),
            concat([A,B,C,D,E],Oword).

  noun_test6([A,B,C,D,E,F],[Oword,RO,DE,SDP,MF,PR,IN,SU])  :-
             nsuf_test6([A,B,C,D,E,F],RO,DE,SDP,MF,PR,IN,SU),
             concat([A,B,C,D,E,F],Oword).
```

```
noun_test6([A,B,C,D,E,F],[Oword,RO,DE,SDP,MF,PR,IN,SU])  :-
          npre_test6([A,B,C,D,E,F],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F],Oword).

noun_test7([A,B,C,D,E,F,G],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          npre_test7([A,B,C,D,E,F,G],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G],Oword).

noun_test7([A,B,C,D,E,F,G],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          nsuf_test7([A,B,C,D,E,F,G],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G],Oword).

noun_test8([A,B,C,D,E,F,G,H],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          npre_test8([A,B,C,D,E,F,G,H],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H],Oword).

noun_test8([A,B,C,D,E,F,G,H],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          nsuf_test8([A,B,C,D,E,F,G,H],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H],Oword).

 noun_test9([A,B,C,D,E,F,G,H,I],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          npre_test9([A,B,C,D,E,F,G,H,I],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H,I],Oword).

noun_test9([A,B,C,D,E,F,G,H,I],[Oword,RO,DE,SDP,MF,PR,IN,SU]) :-
          nsuf_test9([A,B,C,D,E,F,G,H,I],RO,DE,SDP,MF,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H,I],Oword).

noun_testll([A,B,C,D,E,F,G,H,I,J,K],
         [Oword,RO,DE,SDP,MF,PR,IN,SU]) :
         npre_testll([A,B,C,D,E,F,G,H,I,J,K],RO,DE,SDP,MF,PR,IN,SU)
         concat([A,B,C,D,E,F,G,H,I,J,K],Oword).

noun_tstl2([A,B,C,D,E,F,G,H,I,J,K,L],
      [Oword,RO,DE,SDP,MF,PR,IN,SU ]) :
      npre_testl2([A,B,C,D,E,F,G,H,I,J,K,L],RO,DE,SDP,MF,PR,IN,SU),
      concat([A,B,C,D,E,F,G,H,I,J,K,L],Oword).

noun_testl3([A,B,C,D,E,F,G,H,I,J,K,L,M],
     [Oword,RO,DE,SDP,MF,PR,IN, SU]) :
     pre_testl3([A,B,C,D,E,F,G,H,I,J,K,L,M],RO,DE,SDP,MF,PR,IN,SU),
     concat([A,B,C,D,E,F,G,H,I,J,K,L,M],Oword).

npre_test4([A,B,C,D],RO,DE,$مفرد$,$$,PR,$$,$$) :
     member(PR,[$ال$,$ول$,$فل$,$لل$,$وب$,$فب$,$وك$,$فك$]),
     conca(C,D,PR),
     DE = $فل$,
     conca([A,B],RO).
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                        VERB MODULE                          */
/*                                                             */
/*-------------------------------------------------------------*/

  verb_test(L,T,2)  :- verb_test2(L,T).
  verb_test(L,T,3)  :- verb_test3(L,T).
  verb_test(L,T,4)  :- verb_test4(L,T).
  verb_test(L,T,5)  :- verb_test5(L,T).
  verb_test(L,T,6)  :- verb_test6(L,T).
  verb_test(L,T,7)  :- verb_test7(L,T).
  verb_test(L,T,8)  :- verb_test8(L,T).
  verb_test(L,T,9)  :- verb_test9(L,T).
  verb_test(L,T,10)  :- verb_test10(L,T).
  verb_test(L,T,11)  :- verb_test11(L,T).
  verb_test(L,T,12)  :- verb_test12(L,T).

  verb_test2([$هـ$,B],[Oword,R,$ـت$,$ـا$,X,Y,Z,$$,$$,$هـ$])  :-
            member(B,[$ـَ$]),
            concat($هـ$,B,Oword).

   verb_test3([A,B,C],[Oword,Oword,$ـفعـل$,$ـماضـي$,X,Y,Z,$$,$$,$$])  :-
            member(Oword,[$ـنـعـم$,$ـبـئـس$]),
            conca([A,B,C],Oword).

   verb_test3([A,B,C],[Oword,R,$ـهـ$,$ـا$,X,Y,Z,$$,$$,Sfx])  :-
            member(C,[$ـَ$]),
            member(Sfx,[$ـنـي$,$ـهـا$,$ـنـا$,$ـهـم$,$ـهـن$]),
            conca(A,B,Sfx), conca([$ـى$,C,$ـو$],R),
            concat(Sfx,C,Oword).

    verb_test4([A,B,C,D],[Oword,RO,DE,Ty,X,Y,Z,$$,IN,$$])  :-
            vinf_test4([A,B,C,D],RO,DE,Ty,X,Y,Z,IN),
            conca([A,B,C,D],Oword).

   verb_test4([A,B,C,D],[Oword,RO,DE,Ty,X,Y,Z,$$,$$,SU])  :-
            vsuf_test4([A,B,C,D],RO,DE,Ty,X,Y,Z,$$,$$,SU),
            concat([A,B,C,D],Oword).

  verb_test5([A,B,C,D,E],[Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU])  :-
            vpre_test5([A,B,C,D,E],RO,DE,Ty,X,Y,Z,PR,IN,SU),
            concat([A,B,C,D,E],Oword).

   verb_test6([A,B,C,D,E,F],
            [Oword,Oword,$ـفعـل$,$ـماضـي$,X,Y,Z,$$,$$,$$ ])  :-
            conca([A,B,C,D,E,F],Oword), Oword==$ـذاهـب$.

   verb_test7([A,B,C,D,E,F,G],[Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU])  :-
            vpre_test7([A,B,C,D,E,F,G],RO,DE,Ty,X,Y,Z,PR,IN,SU),
            concat([A,B,C,D,E,F,G],Oword).
```

```
verb_test8([A,B,C,D,E,F,G,H],[Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU])  :-
          vsuf_test8([A,B,C,D,E,F,G,H],RO,DE,Ty,X,Y,Z,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H],Oword).

verb_test9([A,B,C,D,E,F,G,H,I],
          [Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU]) :-
          vsuf_test9([A,B,C,D,E,F,G,H,I],RO,DE,Ty,X,Y,Z,PR,IN,SU),
          concat([A,B,C,D,E,F,G,H,I],Oword).

verb_test10([A,B,C,D,E,F,G,H,I,L],
       [Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU]) :-
       vsuf_test10([A,B,C,D,E,F,G,H,I,L],RO,DE,Ty,X,Y,Z,PR,IN,SU),
       concat([A,B,C,D,E,F,G,H,I,L],Oword).

verb_test11([A,B,C,D,E,F,G,H,I,L,J],
       [Oword,RO,DE,Ty,X,Y,Z,PR,IN,SU ])  :-
       vpre_test11([A,B,C,D,E,F,G,H,I,L,J],RO,DE,Ty,X,Y,Z,PR,
       IN,SU), concat([A,B,C,D,E,F,G,H,I,L,J],Oword).

verb_test12([A,B,C,D,E,F,G,H,I,L,J,K],
       [Oword,RO,DE,Ty,X,Y,Z,PR,IN, SU])  :-
       vpre_test12([A,B,C,D,E,F,G,H,I,L,J,K],RO,DE,Ty,X,Y,Z,P
       R,IN,SU), concat([A,B,C,D,E,F,G,H,I,L,J,K],Oword).

vpre_test4([A,B,C,D],RO,DE,TY,$مفرد$,_,$غائب$,D,$$,$$) :-
        member(D,[$ا$,$آ$]),
        conca([A,B,C],RO),
        DE=$فعل$, TY=$ماضي$.

vpre_test5([A,B,$ت$,D,E],RO,DE,$مضارع$,$مفرد$,$مذكر$,$غائب$,E,$ت$,
        $$) :-member(E,[$ي$]),
        DE = $افتعل$,
        conca([A,B,D],RO).

vpre_test5([A,B,C,D,E],RO,DE,TY,$مفرد$,$مذكر$,$غائب$,PR,$$,$$) :-
        member(D,[$ن$,$ت$]),member(E,[$ي$]), conca([A,B,C],RO),
        concat([D,E],PR),TY=$مضارع$, DE = $فعل$.

vsuf_test5([A,B,C,$ا$,E],RO,DE,$ماضي$,X,Y,Z,$$,$$,SU) :-
        conca(A,B,SU), member(SU,[$تم$,$تم$,$وا$,$تك$,
            $كم$,$كم$,$كن$,$ها$,$هم$,$هن$,$نا$,$ني$,$تن$]),
         DE = $فعل$,

vsuf_test10([$ا$,$ه$,$ي$,$ن$,$و$,$م$,$ت$,H,I,L],RO,DE,TY,
          $جمع$,$مذكر$,$مخاطب$,$$,$$,$تمونيها$) :- DE = $فعل$,
          conca([H,I,L],RO), TY=$ماضي$.

vsuf_test11([$ا$,$ه$,$ي$,$ن$,$و$,$م$,$ت$,H,I,L,$آ$],RO,DE,TY,
          $مفرد$,_,$متكلم$,$$,$$,$تمونيها$) :- DE = $فعل$,
          conca([H,I,L],RO), TY=$ماضي$.
```

REFERENCES

1. Al-Authman, A. "A Morhphological Analyzer for Arabic", KFUPM, ICS Dept., M.S.Thesis, 1990.
2. Al-Jabri, S., "A Syntax Analyzer for Arabic", KFUPM, ICS Dept. M.S. Thesis, 1988.
3. Allen, J., "Natural Language Understanding", the Benjamin Publishing Company, California, 1987.
4. Al-Sawadi, A. "An End-Case Analyzer of Arabic Sentences", KFUPM, ICS Dept., M.S. Thesis, 1990.
5. Al-Waer, M., "The Syntactic and Semantic Analysis of the Generation of the Arabic Sentence", Second Conference on Arabic Computational Linguistics, Kuwait, Nov. 1989.
6. Barr, A. and Feignbaum, E., The Handbook of Artificial Intelligence, Vol 1, William Kaufmann, Inc., 1981.
7. Bossie, S. A Tactical Component for Text Generation: Sentence Generation Using a Functional Grammar, Tech. Rep. MS-CIS-81-5, Univ. of pennsylvania, Philadelphia, PA, 1981.
8. Dale, R., Mellish, C. and Zock, M., "Current Research in Natural Language Generation", Univ. of Edinb. and CNRS, P.330, 1990.
9. El-Dessouki, O., "An ATN Approach for Understanding Arabic Sentences", Proceedings of the 11th National Computer Conference, Dhahran, Mar. 1989.

10. Elsharkawi, A. and Hegazi, N., "An Approach to a Computerized Lexical Analyzer for Natural Arabic Text " Computer Processing of the Arabic Language WorkShop, Kuwait, April 1985, vol. 1.

11. Ghith, M. and Aboul.Ela, M. " A Computer Based Arabic Syntax Analyzer", Proceedings of the 11th National Computer Conference, Dhahran, Mar. 1989.

12. Hegazi, N. H. and Elsharkawi, A. "Natural Arabic Language Processing", Proceedings of the 9th national Computer Conference, Riyadh, 1986.

13. Hilal, Y. "Arabic Morphological Generation", Second Conference on Arabic Computational Linguistics, Kuwait, Nov. 1989.

14. Hovy, E., McDonald, D., Young, S., "Current Issues in Natural Language Generation: An Overview of the AAAI Workshop on Text Planning and Realization", AAAI, p. 27, 1989.

15. Khayat, M. "An Arabic Syntax Analyzer", Sabbatical year report, KFUPM, ICS Dept., 1990.

16. Khel, W., "GEOTEX - Ein System zur Verbalisierung Geometrischer Konstruktionen", Diplomarbeit, Institut fur Informatik, Univ. Stuttgart, 1985.

17. Mann, W. and Matthiesse, C. Nigel: A Systemic Grammar For Text Generation. USC Information Sciences Institute Tech. Rep. RR-83-105, 1983.

18. McKeown, K.R. Generating Natural Language Text in Response to Questions About Database Structure. Ph. D. Thesis Univ. of Pennsylvania, 1982.

19 Paris, C., Swartout, W., Mann, W., "Natural Language Generation in Artificial Intelligence and Computational Linguistics", Univ. of South Ca., p. 448, 1991.

20. Rosner, D., "The Automated News Agency: the SEMTEX Text Generator for German", Kluwer Academic Publishers, Dordrecht/Boston, 1987.

21. Rosner, D. "The SEMSYN Generation System: Ingredients, Applications, Prospects", Proceedings of the Second Conference on Applied Natural Language Processing, P. 25, Feb. 1988.