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
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
An Arabic sentence generator

Al-Safran, Safran Ali, M.S.

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

[Signatures]

Thesis Advisor
Member
Member
Member

Department Chairman

Dean, College of Graduate Studies

Date: 4-7-92
To
MY Mother, Father, Brothers, and Sisters
ACKNOWLEDGMENTS

Acknowledgment 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 thesis committee 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: صقران علي يوسف الصقران
عنوان الدراسة: نظام آلي لإنتاج الجمل العربية
التخصص: علوم الحاسب الآلي
تاريخ الدرجة: يوليو 1992 م

يتمثل هذا البحث جزء من "النظام الآلي لفهم اللغة العربية" الذي يتم تطويره في جامعة الملك فهد للبترول والمعادن. ويختص البحث بتطوير نظام إنتاج الجمل العربية.

يقدم هذا البحث تصميم وبنائه نظام آلي لإنتاج الجمل العربية بناء على قواعد النحو والصرف والأعراب. ويعتمد تصميم النظام وبناؤه على تقسيمه إلى أربع وحدات مترابطة هي: وحدة المعالجة الأولية، ووحدة التركيب الصرف، ووحدة التركيب النحوي، ووحدة الأعراب.

وقد تم دمج وحدات التركيب مع وحدات التحليل المقابلة في نظام فهم اللغة العربية لتسهيل عملية بناء المركب الصرفي والتحوي والأعرابي. وقد تم بناء نظام إنتاج الجمل العربية باستخدام لغة بروغ.

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

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>NAUS Components</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>List of Special Article Predicates</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Morphological Module Design</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Sentence Classifications</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Design Structure of the End-Case Module.</td>
<td>27</td>
</tr>
<tr>
<td>2.5</td>
<td>Verb Types</td>
<td>39</td>
</tr>
<tr>
<td>2.6</td>
<td>Verb Irb Categories</td>
<td>40</td>
</tr>
<tr>
<td>2.7</td>
<td>Present Verb Irb Categories</td>
<td>41</td>
</tr>
<tr>
<td>2.8</td>
<td>Article Types</td>
<td>46</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>Verb Derivations</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Noun Derivations</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Sentence End-Case Classification</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>Variable &amp; Invariable Nouns</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Variable Nouns &amp; their End-Case Marks</td>
<td>32</td>
</tr>
<tr>
<td>2.6</td>
<td>Noun Irb Cases &amp; the Corresponding End-Case Positions</td>
<td>35</td>
</tr>
<tr>
<td>2.7</td>
<td>Variable Nouns &amp; their End-Case Marks</td>
<td>36</td>
</tr>
<tr>
<td>2.8</td>
<td>Verb Irb Cases &amp; the Corresponding End-Case Positions</td>
<td>44</td>
</tr>
</tbody>
</table>
CHAPTER 1

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 (الترميم المجمع), is to synthesize a word from the word root and its derivation. This was done in three stages:

1. replace the (قام) of fail of the derivation by the first letter of the root,
2. apply transformation rules that treat the special cases, such as (مصطلح ---> مصطلح), and
3. make a writing correction that handles the letter (همزه), such as (اكل ---> الأكل).

The second method (الترميم النصي) 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 trilab cases (الحالات الأعرابية) (nominative, accusative, and
  genitive),
- for a verb: its tense (past, present, and imperative), its trilab cases
  (indicative, subjunctive, and jussive), and its type (plural, dual,
singular, masculine, 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
التركيب). The last step, is to make writing corrections for some of the
cases, such as (أكلـ (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 \( \text{verb}(X,Y,..) \), where \( X \) is a verb, and \( Y \) is a subject,
- nominal relations are represented by the predicate \( \text{nom}(X,Y,..) \), 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.
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, feminine, masculine), 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.
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 articles followed by a verb, contains at least one article, and contains two or more sentences.
Figure 2.3: Sentence Classifications

Complex Simple

Nominal Verbal Nominal Verbal

Pure Sentence Tools Sentence

Arabic Sentence
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.
<table>
<thead>
<tr>
<th>Sentences with an end-case place</th>
<th>Sentences with no end-case place</th>
</tr>
</thead>
<tbody>
<tr>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
<tr>
<td>וְלֹא יִלְבָּשׁ בָּשָׂם</td>
<td>וְלֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
<tr>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
<tr>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
<tr>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
<tr>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
<td>לֹא יִלְבָּשׁ בָּשָׂם</td>
</tr>
</tbody>
</table>

**TABLE 2.3: Sentence End-Case Classification**

לֹא יִלְבָּשׁ בָּשָׂם
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, (الوضع الإعرابي)، and end-case mark, (العلامات الإعرابية)، 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 ḥaraka 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 (الألف)، 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.
<table>
<thead>
<tr>
<th>End-Case Mark</th>
<th>Nominate</th>
<th>Accusative</th>
<th>Genitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
<tr>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
<td>የጫጎ</td>
</tr>
</tbody>
</table>

TABLE 2.5 : Variable Nouns and their End-Case Marks
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 Ira' cases (الحالات الأعرابية): nominative, accusative, and genitive. A noun can be in one of these cases according to the end case position (الموقع الأعرابي) that it occupies in a sentence. For a complete list of these Ira' cases and the possible end-case positions that cause a noun to be in those Ira' 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.
<table>
<thead>
<tr>
<th>End-Case Position</th>
<th>Nominative</th>
<th>Accusative</th>
<th>Genitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
</tr>
<tr>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
</tr>
<tr>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
</tr>
<tr>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
<td>לְוַעֲרָא</td>
</tr>
</tbody>
</table>

TABLE 2.6: Noun Irbab Cases and the Corresponding End-case Positions

לְוַעֲרָא → לְוַעֲרָא → לְוַעֲרָא → לְוַעֲרָא

35
<table>
<thead>
<tr>
<th>End-Case Mark</th>
<th>Nom. Type</th>
<th>Acc. Type</th>
<th>Gen. Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
</tr>
<tr>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
</tr>
<tr>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
<td>✴️</td>
</tr>
</tbody>
</table>

**Table 2.7: Variable Nouns and Their End-Case Marks**
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 التوكل) or (دنون النسوة).

A variable verb suffers from changes in its end-case mark according to changes in its Iraab case in the sentence. Variable verbs have three Iraab 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 (حرف النصب). If it is preceded by one of the jussive
articles, (الحروف 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 ird Categories
Figure 2.7 : Present Verb Irbab 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 Arab 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 Irb 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.
<table>
<thead>
<tr>
<th>End-case positions</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb inflection</td>
<td>Indicative</td>
<td>Subjunctive</td>
<td>Jussive</td>
</tr>
<tr>
<td>End-case Mark</td>
<td>4th</td>
<td>6th</td>
<td>7th</td>
</tr>
</tbody>
</table>

TABLE 2.8: Verb Inflection & The Corresponding End-case Mark
**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 عامل) 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 (applicable 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 (العلامات الإعرابية), 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, syntactic, 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 assosciativity 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 embedded 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 \textit{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 \textit{conca}(A,B,C), as shown below, that performs these two functions.

\[
\text{CONCA}(A,B,C) :- \text{concat}(A,B,C); \text{word_lst}(C,[A,B]).
\]

The function of the predicate \textit{word_lst}(C,[A,B]) is to break a word represented by the variable C into two parts A and B.

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

As an example of the replacement of \textit{concat} by \textit{conca}, the predicate before the replacement takes place is:

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

\[
\text{member}(D,["ت","ن","لا","م"],\text{concat}([A,B,C],RO),
\]

\[
\text{concat}("\text{ فعل }",D],DE), TY="\text{ مضارع }", PR=D.
\]

The predicate after the replacement takes place is:

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

\[
\text{member}(D,["ت","ن","لا","م"],\text{conca}([A,B,C],RO),
\]

\[
\text{conca}("\text{ فعل }",D],DE), TY="\text{ مضارع }", PR=D.
\]
2- Delaying the call to the predicate \texttt{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

\begin{verbatim}
vpre_test6([A,B,C,D,E,F],RO,DE,TY,PR,"","",":"):-
    concat(E,F,P),
    member(P,["ن","س","ر","ص","ك","ل","ل"]),
    member(D,["ن","ي","ت","ت","ت","ت","ت"]),
    concat([A,B,C],RO), concat("فعل",[D,P],DE),
    concat([D,P],PR), TY="مضياع".
\end{verbatim}

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 \texttt{conca}. Binding the variable P can be done by delaying the call to the predicate \texttt{conca} after the predicate \texttt{member}. After performing the change, the predicate is as listed below

\begin{verbatim}
vpre_test6([A,B,C,D,E,F],RO,DE,TY,PR,"","",":"):-
    member(P,["ن","س","ر","ص","ك","ل","ل"]),
    member(D,["ن","ي","ت","ت","ت","ت","ت"]), concat(E,F,P),
    concat([A,B,C],RO), DE="فعل" concat([D,P],PR), TY="مضياع".
\end{verbatim}

In the analysis mode, the predicate \texttt{member}(P,["ن","س","ر","ص","ك","ل","ل"]) checks if the variable P is part of the list ["ن","س","ر","ص","ك","ل","ل"] or not. While in synthesis, it assigns to the variable P one of the values that is in the list ["ن","س","ر","ص","ك","ل","ل"]. 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

\[
\text{verb}\_\text{test5}([A,B,C,D,E], T):-
\]

\[
\text{vpre}\_\text{test5}([A,B,C,D,E], RO, DE, Ty, Agender, Number, PR, IN, SU),
\]

\[
\text{concat}([A,B,C,D,E], O\text{word}),
\]

\[
T=[O\text{word}, R, DE, Ty, Agender, Number, PR, IN, SU].
\]

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:

\[
\text{verb}\_\text{test5}([A,B,C,D,E],
\]

\[
[O\text{word}, R, DE, Ty, Agender, Number, PR, IN, SU]):-
\]

\[
\text{vpre}\_\text{test5}([A,B,C,D,E], RO, DE, Ty, PR, IN, SU),
\]

\[
\text{concat}([A,B,C,D,E], O\text{word}),
\]

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 \(\text{concat}(\_\_\_\_\_\_\_)\) by the predicate \(\text{conca}(\_\_\_\_\_\_\_)\), for the same purpose as mentioned above in section 3.1.1.

As an example of the replacement of \(\text{concat}\) by \(\text{conca}\), the predicate before the replacement takes place is:
\[ npre\_test4([A,B,C,D],RO,DE,"","",D,"","") :- \]
\[ member(D,["","","","",L","",M,"",F,"",K,"",B,"",P,"","","","","","])), \]
\[ \text{concat([A,B,C],RO), \text{concat("","",D],DE).} \]

The predicate after the replacement takes place is:

\[ npre\_test4([A,B,C,D],RO,DE,"","","",D,"","") :- \]
\[ member(D,["",L","",M,"",F,"",K,"",B,"",P,"","","","","])), \]
\[ \text{concat([A,B,C],RO),} \]
\[ \text{concat("",D],DE).} \]

2- Delaying the call to the predicate \text{concat(E,F,P)} in some predicates, for the same purpose as mentioned above in 3.1.1.

For example, the predicate before performing the change is,

\[ npre\_test7([A,B,C,D,E,F,G],RO,DE,"",",",PR,IN,SU) :- \]
\[ \text{concat(F,G,M),} \]
\[ \text{member(M,["",",",",",",",",",",",",",",",",",",","])),} \]
\[ \text{concat([A,B,C,D,E],RO,DEE,SDP,MF,PRE,IN,SU),} \]
\[ \text{concat(PRE,M,PR),} \]
\[ \text{concat(DEE,M,DE).} \]

After the modification, it is changed to

\[ npre\_test7([A,B,C,D,E,F,G],RO,DE,"",",",PR,IN,SU) :- \]
\[ \text{member(M,["",",",",",",",",",",",",",",",",","])),} \]
\[ \text{concat(F,G,M),} \]
\[ \text{concat([A,B,C,D,E],RO,DE,SDP,MF,PRE,IN,SU),} \]
\[ \text{concat(PRE,M,PR),} \]
\[ \text{concat(DEE,M,DE).} \]
concat(PRE,M,PR).

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

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

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

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

\textbf{3.2.3 The Article Submodule}

The changes that made in this sub-module are:

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

As an example of the replacement of \texttt{concat} by \texttt{conca}, the predicate before the replacement takes place is:

\[
\text{find_art2([A,B],T,Root,Type,B,\\"\\""):-}
\]
\[
\text{concat([A,B],Root),}
\]
\[
\text{member(Root,"\"ان\","ي")},
\]
\[
\text{Type = "حرف تفسير", T=2.}
\]

After the modification it is changed to

\[
\text{find_art2([A,B],T,Root,Type,B,\\"\\""):-}
\]
conca([A,B],Root),
member(Root,[["ان","\"اي\""],[" 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:

\[ \text{art\_test2([A,B],T):- find\_art2([A,B],TC,RO,Type,PR,SU), } \]
\[ \text{concat([A,B],Oword), } \]
\[ T=[Oword,TC,RO,Type,PR,SU]. \]

After the modification, it is changed to

\[ \text{art\_test2([A,B],[Oword,TC,RO,Type,PR,SU]):- find\_art2([A,B],TC,RO,Type,PR,SU), concat([A,B],Oword). } \]
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 \textit{substring}(X,a,b,\textit{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 bound during 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

\[
esm\_marfo(X0,[X1,_,X3,$\text{ اسم}$,$\text{ مؤتث}$,$\text{ Part}$,Out):-
\text{substring}(X1,0,2,\text{Part}), \text{ Part} == \text{اًت},
\text{Out} =$\text{ اسم$ مؤتث$}$,
\text{rite}([\text{Out},';','X0',';',X1]),nl.
\]

is modified to
\texttt{esm\_marfo(X0,[X1,_X3,$\textasciitilde$,,$\textasciitilde$,\textasciitilde$,Part] ,Out):-}
\texttt{ifthen((\textasciitilde+\texttt{var(X1)}),(\texttt{substring(X1,0,2,Part)}))},
\texttt{Part ==$\textasciitilde$},
\texttt{Out =$\textasciitilde$}.
\texttt{\texttt{rite([Out,'\textasciitilde',X0,'\textasciitilde',X1]),nl.}}

In the synthesis mode the variable Part will be assigned a value by the statement Part ==$\textasciitilde$, 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 Compound
- Non-Article Verbal Compound
- Article Nominal Compound
- Article Verbal Compound.

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, suffix, and type) in all the predicates before using them in the synthesis mode. For example, in the following predicate

\[
\text{esm_eshara}(L,Ninfo,[Jns,\text{Hal}]) \rightarrow
\]

\[
\text{length}(L,N),
\]
\[
\text{art_test}(L,[\text{Wrd},\text{Rt},\text{De}]),
\]
\[
\text{hal_jns}(\text{De},\text{Jns},\text{Hal}),
\]
\[
Ninfo=[\text{Wrd},\text{Rt},\text{De},\text{Jns},\text{Hal}].
\]
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:

\[\text{esm_eshara}(L,[\text{Wrd}, \text{Rt}, \text{De}, @\text{Ninfo}@], \text{Jns}, \text{Hal}, @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx], \text{Jns}, \text{Hal}, @\text{Mعرفه}@)\]

\[\text{length}(L,N),\]

\[\text{art_test}(L,[\text{Wrd}, \text{Rt}, \text{De}, @\text{Ninfo}@], @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx], N)]\]

\[\text{hal_ins}(\text{De}, \text{Jns}, \text{Hal})\]

2- The predicate \text{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 \text{length}(L,N) has the variable L bound, however, during the synthesis mode both variables L, and N are unbound variables. For example, the following predicate

\[\text{esm_eshara}(L,[\text{Wrd}, \text{Rt}, \text{De}, @\text{Ninfo}@], \text{Jns}, \text{Hal}, @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx], \text{Jns}, \text{Hal}, @\text{Mعرفه}@)\]

\[\text{length}(L,N),\]

\[\text{art_test}(L,[\text{Wrd}, \text{Rt}, \text{De}, @\text{Ninfo}@], @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx], N)]\]

\[\text{hal_ins}(\text{De}, \text{Jns}, \text{Hal})\]

is modified to:

\[\text{esm_eshara}(L,[\text{Wrd}, \text{Rt}, \text{De}, @\text{Ninfo}@], \text{Jns}, \text{Hal}, @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx], \text{Jns}, \text{Hal}, @\text{Mعرفه}@, @\text{Mعرفه}@, @\text{Mعرفه}@, Sfx]
[Jns, Hal, $]::-
ifthen( l+ var(L)), (length(L, N)),
art_test(L, [Wrd, Rt, De, $, $$$, 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

\[
\text{bfb_feil_2}([L1|L2],[Nom1,Nom2,Nom3],Rem):-
\]

\[
[\text{ileil}(L1,[\text{Type}_1],Nom1,Lsfx1)|],
\]

\[
\text{fael}(Lsfx1,L2,Nom2,Jhmn2,L3,[$$]),
\]

\[
\text{tbi_fli}(L3,Nom3,Jhmn3,Rem),
\]

\[
\]

is modified to:
During the analysis mode the program will skip the first end-case calls by checking the condition \texttt{var(L2)} which will be false during this stage. However, in synthesis mode the condition \texttt{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:

\[
\text{verb}(\text{feil}([A,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$,\$
4- The syntactic module calls the end-case module with the following predicate:

\[ \text{asma_marfo}([\text{اسم_i}, [\text{D}, \text{F}]]) \]

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:

\[ \text{asma_marfo}([\text{اسم_i}, [\text{D}, \text{F}]]) \]

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

\[ \text{noun_test}([\text{اسم_i}, [\text{D}, \text{F}]] \]

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

\[ \text{noun_test}([\text{اسم_i}, [\text{D}, \text{F}]] \]

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

\[ \text{asma_mnsb}([\text{اسم_i}, [\text{G}, \text{H}]]) \]

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:

\[ \text{asma_mnsb}([\text{اسم_i}, [\text{G}, \text{H}]]) \]

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

\[ \text{noun_test}([\text{اسم_i}, [\text{G}, \text{H}]]) \]

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

\[ \text{noun_test}([\text{اسم_i}, [\text{G}, \text{H}]]) \]
8- The output of the syntactic module is:

\[
\text{bfb (عملية بسيطة)}\text{،}
\]

\[
[\text{عملية بسيطة،}]
\]

\[
[\text{عملية بسيطة،}]
\]

\[
[\text{عملية بسيطة،}]
\]

9- The syntactic module produces the sentence "قاتل المسلمين الكفار".
EXAMPLE 2

The input to the preprocessor module is:

\[
\text{ma_t}(\text{verb}(\text{feil}([A,$\$,B,$\$,C,$\$,D]), \text{mfol}([\text{E,$\$,F,$\$,G}])))
\]

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 "معفعل به". The preprocessor calls the syntactic module by the following predicate:

\[
\text{afg}(L,[X,[\text{جملة تعجب}],[H,2,\text{اسم مرفوع}],[1,1])]
\]

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

\[
\text{esm_marfo}([\text{جملة تعجب}],[H,2,\text{اسم مرفوع}])
\]

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

\[
\text{esm_marfo}([\text{Name}],[H,2,\text{اسم مرفوع}])
\]

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

\[
\text{art_test}(X,[H,2,\text{اسم مرفوع}])
\]

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

\[
\text{art_test}([\text{ما}],[1,2,\text{اسم مرفوع}])
\]

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

\[
\text{feil_am}([\text{Name}],[A,$\$,B,$\$,C,$\$,D])
\]
The end-case module determines the prefix "I" of the word, and it calls the syntactic module by the following predicate with the new parameters:
\[ \text{tell}_\text{am}(\$_1,\$_2,\$_3,\$_4,\$_5,\$_6,\$_7,\$_8,\$_9). \]

5- The syntactic module calls the morphological module with the following predicate:
\[ \text{verb}_\text{test}(L,\$_1,\$_2,\$_3,\$_4,\$_5,\$_6,\$_7,\$_8,\$_9,\$_{10}). \]

The morphological module constructs the word "بـأجمل", finds the length of the word "4", and it calls the syntactic module with the following predicate:
\[ \text{verb}_\text{test}(\$_{11},\$_{12},\$_{13},\$_{14},\$_{15},\$_{16},\$_{17},\$_{18},\$_{19},\$_{20},\$_{21}). \]

6- The syntactic module calls the end-case module with the following predicate:
\[ \text{asma}_\text{mnssb}(\$_{22},\$_{23},\$_{24},\$_{25},\$_{26},\$_{27},\$_{28},\$_{29}). \]

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:
\[ \text{asma}_\text{mnssb}(\$_{30},\$_{31},\$_{32},\$_{33},\$_{34},\$_{35},\$_{36},\$_{37},\$_{38},\$_{39},\$_{40}). \]

7- The syntactic module calls the morphological module with the following predicate:
\[ \text{noun}_\text{test}(L,\$_{41},\$_{42},\$_{43},\$_{44},\$_{45},\$_{46},\$_{47},\$_{48},\$_{49},\$_{50}). \]

The morphological module constructs the word "البدرس", finds the length of the word "5", and it calls the syntactic module with the following predicate:
\[ \text{noun}_\text{test}(\$_{51},\$_{52},\$_{53},\$_{54},\$_{55},\$_{56},\$_{57},\$_{58},\$_{59},\$_{60}). \]

8- The output of the syntactic module is:
\[ \text{afb}(\$_{61},\$_{62},\$_{63},\$_{64},\$_{65},\$_{66},\$_{67},\$_{68},\$_{69}). \]

9- The syntactic module produces the sentence "ما أجمل البدرس".
EXAMPLE 3

The input to the preprocessor module is:

\[
\text{inna}_n(\text{nom}([A,B,C]), \text{kmbr}([D,E,F,G])))
\]

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 sentence "حرف ناسخ", type of the first construct "دريازة اسمية بسيطة", type of the second construct "ميتشا", and the type of the last construct "خبر". The preprocessor calls the syntactic module by the following predicate:

\[
\text{art}_\text{test}(L,[A,B,C,D,E,F,G,N]).
\]

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

\[
\text{art}_\text{test}([\ldots],[A,B,C,D,E,F,G,N]).
\]

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

\[
\text{asma}_\text{mnsb}([\ldots],[B,C,D,E,F,G,N]).
\]

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

\[
\text{asma}_\text{mnsb}([\ldots],[B,C,D,E,F,G,N]).
\]

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

\[
\text{noun}_\text{test}(X,[L],[B,C,D,E,F,G,N]).
\]
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([[فعلی]]).
```

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

```
asma_marfo(اجماع,$\text{کریم}$,[E,$\text{کریم}$,$\text{نکته}$,$\text{ضمن}$,$\text{نمی}$,$\text{ست}$,$\text{مکرر}$,$\text{مفرد}$,$\text{نکته}$,$\text{کریم}$,$F$,$\text{فی}$,$G$])
```

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

```
asma_marfo(اجماع,$\text{کریم}$,[E,$\text{کریم}$,$\text{نکته}$,$\text{ضمن}$,$\text{نمی}$,$\text{ست}$,$\text{مکرر}$,$\text{مفرد}$,$\text{نکته}$,$\text{کریم}$,$\text{فی}$,$\text{فی}$])
```

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

```
noun_test(X,[E,$\text{کریم}$,$\text{نکته}$,$\text{ضمن}$,$\text{نمی}$,$\text{ست}$,$\text{مکرر}$,$\text{مفرد}$,$\text{نکته}$,$\text{کریم}$,$\text{فی}$,$\text{فی}$]).
```

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

```
noun_test([\text{کریم}],[\text{کریم}$,$\text{نکته}$,$\text{ضمن}$,$\text{نمی}$,$\text{ست}$,$\text{مکرر}$,$\text{مفرد}$,$\text{نکته}$,$\text{کریم}$,$\text{فی}$,$\text{فی}$,4]).
```

7- The output of the syntactic module is:

```
bfb(L,[[\text{کریم}],[\text{کریم}$,$\text{نکته}$,$\text{ضمن}$,$\text{نمی}$,$\text{ست}$,$\text{مکرر}$,$\text{مفرد}$,$\text{نکته}$,$\text{کریم}$,$\text{فی}$,$\text{فی}$]]).
```

8- The syntactic module produces the sentence "آن الرجل کریم". 
EXAMPLE 4

The input to the preprocessor is:

```plaintext
nom(mbtd([X],[A,$B,C]),
    khbr(verb(fail([D,$E,F]),
            fail([G,$L,M]),
            mfol([N,$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 "Siutta" and the type of the second construct "سيدة". The preprocessor calls the syntactic module by the following predicate:

```plaintext
bfm([X],[A,$B,C]),
    [D,$E,F],
    [G,$L,M],
    [N,$O,P]]).
```

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

```plaintext
asma_marfo([X],[A,$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:

```plaintext
asma_marfo([X],[A,$B,C]).
```

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

```plaintext
noun_test([X],[A,$B,C]).
```

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

```plaintext
noun_test([X],[A,$B,C]).
```
4- The syntactic module calls the end-case module with the following predicate:
\[ feil\_am(\text{[D},\text{S},\text{E},\text{F}]\). \]

The end-case module determines the suffix "ت" of the word, and it calls the syntactic module by the following predicate with the new parameters:
\[ feil\_am(\text{[D},\text{S},\text{F}]\). \]

5- The syntactic module calls the morphological module with the following predicate:
\[ verb\_test(X,\text{[D},\text{S},\text{F},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,\text{[D},\text{S},\text{F},N])\).

6- The syntactic module calls the end-case module with the following predicate:
\[ asma\_marfo(\text{[G},\text{L},\text{M}])\).

The end-case module calls the syntactic module by the following predicate with the new parameters:
\[ asma\_marfo(\text{[G},\text{L},\text{M}])\).

7- The syntactic module calls the end-case module with the following predicate:
\[ asma\_mnsb(\text{[N},\text{O},\text{F}])\).

The end-case module calls the syntactic module by the following predicate with the new parameters:
\[ asma\_mnsb(\text{[N},\text{O},\text{F}])\).
8- The output of the syntactic module is:

```
bim([( arabic word ) , arabic word , arabic word , arabic word ]).
```

9- The syntactic module produces the sentence "الرجل دعوته".
EXAMPLE 5

The input to the preprocessor module is:
\[\text{nom(mbtd([A, B, C])],}\]
\[\text{khbr(verb(fail([D, E, F])],}\]
\[\text{fael([O, P, G, H, I, L, K]),}\]
\[\text{mfal([L, M, 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 "جملة فعلية"، and the type of the second construct "مثبتا بسيطة". The preprocessor calls the syntactic module by the following predicate:
\[\text{bam([X, [I, $\ldots$], [A, B, C]])],}\]
\[\text{[O, P, G, H, I, L, K]]}]
\[\text{[L, M, N], Rem}]

2- The syntactic module calls the end-case module with the following predicate:
\[\text{asma_marfo([I, $\ldots$], [A, B, C])}]\]

The end-case module determines the prefix "إ" of the word, and it calls the syntactic module by the following predicate with the new parameters:
\[\text{asma_marfo([I, $\ldots$], [A, B, C])}]\]

3- The syntactic module calls the morphological module with the following predicate:
\[\text{noun_test([X, A, B, C])}]\]

The morphological module constructs the word "الرجل"，find the length of the word "5"，and it calls the syntactic module with the following predicate:
\[\text{noun_test([5, A, B, C])}]\]
4- The syntactic module calls the end-case module with the following predicate:
\[ \text{feil\_am}(D,\text{ناحیه},\text{مکتب},\text{ضریب},\text{شماره},E,\text{سربلاک},\text{نفرات},\text{شماره},F,\_). \]

The end-case module determines the prefix "پ" of the word, and it calls the syntactic module by the following predicate with the new parameters:
\[ \text{feil\_am}(D,\text{ناحیه},\text{مکتب},\text{ضریب},\text{شماره},\text{شماره},\text{شماره},\_). \]

5- The syntactic module calls the morphological module with the following predicate:
\[ \text{verb\_test}(X,\text{ناحیه},\text{مکتب},\text{ضریب},\text{شماره},\text{شماره},\text{شماره},\_). \]

The morphological module constructs the word "پرک", find the length of the word "4", and it calls the syntactic module with the following predicate:
\[ \text{verb\_test}([\text{پرک},\text{شماره},\text{ضریب},\text{شماره},\text{شماره},\text{شماره},\text{شماره},\_],4). \]

6- The syntactic module calls the end-case module with the following predicate:
\[ \text{asma\_marfo}(\text{ناحیه},\text{مکتب},\text{ضریب},\text{شماره},\text{شماره},\text{شماره},\text{شماره},\_). \]

The end-case module calls the syntactic module by the following predicate with the new parameters:
\[ \text{asma\_marfo}([\text{شماره},\text{ضریب},\text{شماره},\text{شماره},\text{شماره},\_]). \]

7- The syntactic module calls the end-case module with the following predicate:
\[ \text{asma\_mnsb}(\text{ناحیه},\_). \]

The end-case module determines the suffix "ت", prefix "ل" of the word, and it calls the syntactic module by the following predicate with the new parameters:
\[ \text{asma\_mnsb}([\text{سربلاک},\_]). \]

8- The syntactic module calls the morphological module with the following predicate:
\[ \text{noun\_test}(X,\_). \]
The morphological module constructs the word "السيارة", find the length of the word "7", and it calls the syntactic module with the following predicate:

\[ \text{noun\_test}(\text{سيارة}, \text{7}) \]

9- The output of the syntactic module is:

\[ \text{bam}(\text{الرجل يركب السيارة}, \text{7}) \]

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 synthesis) 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
start:
   /* 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(K,|],L)!!.

   ![ go_throught_the_list(R,C,T2) :-
   instance(R,T),
   ifthenelse((T==bfb;T==$فرگ$;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==$فرگ$;T==$わからない$),([T,D]|=[T,C], T2=[T,D], T1=[T2])!),
   ifthen((T=bfb,[A,B,D]=C),([T2|=$كلمة سريعة$,[A,B,E]],D,T1=T2)),
   ifthen((T=bfb, var(T2)), (([A,B]=C,T2=[$كلمة سريعة$,[A,B]], T1=T2))!,
               (A,B,D]=C,T2=[|=$كلمة سريعة$,[A,B,D]], T1=T2)),
   ifthen((T==|$كلمة سريعة$),([A,B,D,E]=C,T2=[|=$كلمة سريعة$,[A,B,D]], T1=T2)),
   ifthen((T==nan), (add(C,T,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=[|=$كلمة سريعة$,[A,B,E,F]], T1=T2)).

syntactic_calls(Flag,T,[T1]) :-
   Flag=t, (T==|$كلمة سريعة$), 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]]))
               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,$$ش<$,A), X,Y.
nom(X,Y) :-
             recorda(1,nnn,A1), X,Y.
nom(X,Y,Z) :-
             recorda(1,nnn,A1), X,Y,Z.

feil(X) :- X=[_]:_, Y=[$ فعل,X], recorda(1,Y,B8).
fael(X) :- X=[_]:_, Y=[$فعل+,X], recorda(1,Y,Bbb1).
fael(X) :- recorda(1,$$فعل, B6), X,
             recorda(1,$$فعل, B9).
mbtd(X) :- X=[_]:_, Y=[$شدة$,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=[$ملف$,X], recorda(1,Y,Bb3).

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

jr_mjror(X) :- recorda(1,$$عموم ومحرر$,X), B10.
esm_nkra(X) :- recorda(1,$$اسم نكرة$,X), B1).
bab(L,[{},$\text{اسم سيرة}$,Nom],[$\text{سفر}$,Pred],Rem) :-
mbtda(L,Nom,[Jns1, Hall, _], Rem),
khbr(Rem1, Pred, [Jns2, Hall2, _], Rem),
mtchh([Jns1, Hall], [Jns2, Hall2], [Jns, Hall]).

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,[{}$\text{اسم}$ نكرة$\}$,
[Wr1, Rt, De, @$\text{اسم}$ Ns, Hall, @$\text{نكرة}$ Prfx, Infx, Sfx],
[Jns, Hall, @$\text{نكرة}$]) :-
ifthen(( "\ + var(L) " , (length(L,N))) ,
noun_test(L, [Jns, Hall], Prfx, Infx, Sfx], N),
+ member(Prfx, \$\text{اسم}$ Prfx, Infx, Sfx], N),
$\text{اسم}$ NFs, Ns, Hall, @$\text{نكرة}$, Sfx], N),

esm_eshara(L,[{}$\text{اسم}$ اشارة$\}$,
[Wr1, Rt, De, @$\text{اشارة}$ Ns, Hall, @$\text{معرفة}$ Prfx, Infx, Sfx],
[Jns, Hall, @$\text{معرفة}$]) :-
+ ifthen(( "\ + var(L) " , (length(L,N))) ,
ar_test(L, [Wr1, Rt, De, @$\text{اسم}$ اشارة$\}$, Sfx], N)]),
hal_jns(De, Jns, Hall).

esm_al(L,[{}$\text{اسم}$ علم$\}$,
[Wr1, Rt, De, @$\text{اسم}$ Ns, Hall, @$\text{معرفة}$ Prfx, Infx, Su],
[Jns, Hall, @$\text{معرفة}$]) :-
ifthen("\ + (var(L)) " , (length(L,N))) ,
noun_test(L, [Wr1, Rt, De, Hall, Ns, Prfx, Infx, Su], N),
member(Prfx, \$\text{اسم}$ Prfx, Infx, Su], N),

esm_al(L,[{}$\text{اسم}$ علم$\}$,
[Wr1, Wrd, Wrd, @$\text{اسم}$ Ns, Hall, @$\text{معرفة}$ Prfx, Infx, Su],
[Jns, Hall, @$\text{معرفة}$]) :-
+ lst_word(L, Wrd), alm(Wrd, [Jns, Hall])!].

dameer(L,[{}$\text{ضمير}$ منفّض$\}$,
[Wr1, Rt, De, @$\text{ضمير}$ منفّض$\}$, Ns, Hall, @$\text{معرفة}$ Prfx, Infx, Sfx],
[Jns, Hall, @$\text{معرفة}$]) :-
+ ifthen(( "\ + var(L) " , (length(L,N))) ,
ar_test(L, [Wr1, Rt, De, @$\text{ضمير}$ منفّض$\}$, Sfx], N),
hal_jns(De, Jns, Hall)!].
bfb([11;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([11;L2],[Nom1,Nom2,Nom3,Nom4],Rem) :-
[[!feil(L1,[Typ,|ـ ن مل$],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),

bfb_feil([11;L2],[Nom1,Nom2,Nom3,Nom4,Nom5],Rem) :-
[[!feil(L1,[Typ,|ـ ن مل$],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([11;L2],[Nom1,Nom2,Nom3,Nom4,Nom5,Nom6],Rem) :-
[[!feil(L1,[Typ,|ـ ن مل$],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,[$$]),
 tbi_fli(L6,Nom6,Jhmn6,Rem),
 [X,Nom]=Nom2, asma_marfo(X,Nom),
 [M1,Nom7]=Nom3,
 asma_mnsb(M1,Nom7),
 [M2,Nom77]=Nom4,
 asma_mnsb(M2,Nom77),
 [M3,Nom777]=Nom5,
 asma_mnsb(M3,Nom777).
bfb_feil([L1,L2], [Nom1,Nom2,Nom3], Rem) :-
    ![feil(L1,[Typ,$$\text{متعدد}$],Nom1,Lsf1)!],
    naeb_feil(Lsf1,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,$$\text{متعدد2}$],Nom1,Lsf1)!],
    naeb_fael(Lsf1,L2,Nom2,Jhmn2,L3,Lsf2),
    mfoi(Lsf2,L3,Nom3,Jhmn3,L4,[$$]),
    tbi_fli(L4,Nom4,Jhmn4,Rem),
    [X,Nom]=Nom2,
    asma_marfo(X,Nom),
    [M1,Nom7]=Nom3,
    asma_mnsb(M1,Nom7).

feil(L,[Typ,Lm],[$$,\text{نوع}$,Finfo],Lsf) :-
    ifthen( (\+ var(L)),
         (length(L,N)) ),
    verb_test(L,Finfo,N),
    xtrct(10,Finfo,Sfx),
    word_lst(Sfx,Lsf),
    xtrct(4,Finfo,Typ).

fael(Lsf,L,[$,\text{ضمير}$,Nom],Jhmn,Rem,Lsf) :-
    ![khbr(L,Nom,Jhmn,Rem)!].

fael(Lsf,L,[$,\text{ضمير}$,[$$,\text{معركة}$,L,Jhmn,Rem,Lsf] :-
    lst_word(Lsf,Sfx),
    member(Sfx,
    [$$, $$$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $$, $}]}.

fael(Lsf,L,[$,\text{ضمير}$,Nom],Jhmn,Rem,Lsf) :-
    ![khbr(L,Nom,Jhmn,Rem)!].

mfoi($$$,L,[$,\text{ضمير}$,Nom],Jhmn,Rem,_) :-
    khbr(L,Nom,Jhmn,Rem).
aab(L, [{$میتذ,$Nom}, [$خبر,$Pred]], Rem) :-
mtbda(L, Nom, [Jns1, Hall, _, Rem]),
khbr_dwt2(Rem, Pred, [Jns2, Hal2, _, Rem]),
mtchjh([Jns1, Hall], [Jns2, Hal2], [Jns, Hall]).

aab(L, [{$میتذ,$Nom}, [$خبر,$Pred]], Rem) :-
mtb_dwt2(L, Nom, [Jns1, Hall, _, Rem]),
khbr_dwt1(Rem, Pred, [Jns2, Hal2, _, Rem]),
mtchjh([Jns1, Hall], [Jns2, Hal2], [Jns, Hall]).

aab(L, [{$خبر,$Pred}, [$میتذ,$Nom]], Rem) :-
khbr_dwt2(L, Pred, [Jns1, Hall, _, Rem]),
khbr_dwt1(Rem, Nom, [Jns2, Hal2, _, Rem]),
mtchjh([Jns1, Hall], [Jns2, Hal2], [Jns, Hall]).

aab(L, [{$خبر,$Pred}, [$میتذ,$Nom]], Rem) :-
kbr_dwt2(L, Pred, [Jns1, Hall, _, Rem]),
mtb_dwt1(Rem, Nom, [Jns2, Hal2, _, Rem]),
mtchjh([Jns1, Hall], [Jns2, Hal2], [Jns, Hall]).

aab_mwkda([L1:L2], [Nom1, Nom2], Rem) :-
   [linna(L1, Nom1, [{$!}]!)],
   ifthen( (var(L2)),
   [_, [Y1, Y2], [Z1, Z2]] = Nom2,
   asma_mnsb($سن,$Y2),
   asma_marfo($خبر,$Z2),
   Nom2=[_, [Y1, Y2], [Z1, Z2]])
   bab(L2, Nom2, Rem),
   adat(Nom1, _),
   [_, [Y1, Y2], [Z1, Z2]] = Nom2,
   asma_mnsb($سن,$Y2),
   asma_marfo($خبر,$Z2).

mtb_dwt1(L, Nom, Jhmn, Rem) :-
mtbda(L, Nom, Jhmn, Rem).

mtb_dwt2(L, Nom, Jhmn, Rem) :-
mtb_dwt2(L, Nom, Jhmn, Rem).

jr_mjror([L1:L4], [{$میتذ و مشروو,$Nom1, Nom2}], Rem) :-
   [hrf_jr(L1, Nom1, L3),
   ifthenelse((L3=[{$!}]), (L2=L4), (L2=[L3:L4])!],
   ifthen( (var(L4)), (asma_mjror($حساس محور,$Nom2)) ),
   mtbda(L2, Nom2, Jhmn2, Rem),
   adat(Nom1, _),
   asma_mjror($حساس محور,$Nom2).

eghra_dwt(L, [Nom1, Nom2, Nom3], Rem) :-
   [mtb_dwt1(L, Nom1, Jhmn, L2),
   hrf_waw(L2, Nom2, L3)!],
   ifthen( (var(L2)), (asma_mnsb($معروف,$Nom3)) ),
   mtb_dwt3(L2, Nom3, Jhmn, Rem),
   asma_mnsb($معروف به متهد محور,$Nom1),
   adat(Nom2, _),
   asma_mnsb($معروف,$Nom3).

aab_esthm([L1:L2], [{$میتذ,$Nom1}, [$خبر,$Nom2]], Rem) :-
ifthen( (var(L2)), (asma_marfo($خبر,$Nom1)) ),
asma_esthm($خبر,$L1, Nom1),
ifthen( (var(L2)), (asma_marfo($خبر,$Nom2)) ),
kbr_dwt2(L2, Nom2, Jhmn, Rem),
asma_marfo($خبر,$Nom1), asma_marfo($خبر,$Nom2).
```plaintext
afb([L1:L2], [Nom1,Nom2,Nom3], Rem) :-
    [ifeil(L1,[Typ,|],Nom1,Lsf xl)],
    ifthen((var(L2),
        ([X,Nom]=Nom2, asma_marfo(X,Nom),Nom2=[X,Nom]),
        fael_adwt(Lsf xl,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], [Nom1,Nom2,Nom3,Nom4], Rem):-
    [ifeil(L1,[Typ,|],Nom1,Lsf xl)],
    ifthen((var(L2),
        ([X,Nom]=Nom2, asma_marfo(X,Nom),Nom2=[X,Nom],
            [M,Nom7]=Nom3, asma_mnsb(M,Nom7),
            Nom3=[M,Nom7]),
        fael_adwt(Lsf xl,L2,Nom2,Jhmn,L3,Lsf x2),
        mfol_adwt(Lsf x2,L3,Nom3,Jhmn3,L4,|$$|),
        mfol_adwt(Lsf x3,L4,Nom4,Jhmn4,L5,|$$|),
        tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
        [X,Nom]=Nom2, asma_marfo(X,Nom),

afb([L1:L2], [Nom1,Nom2,Nom3,Nom4,Nom5], Rem) :-
    [ifeil(L1,[Typ,|],Nom1,Lsf xl)],
    fael_adwt(Lsf xl,L2,Nom2,Jhmn2,L3,Lsf x2),
    mfol_adwt(Lsf x2,L3,Nom3,Jhmn3,L4,Lsf x3),
    mfol_adwt(Lsf x3,L4,Nom4,Jhmn4,L5,Lsf x4),
    mfol_adwt(Lsf x4,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).

afb([L1:L2], [Nom1,Nom2,Nom3,Nom4,Nom5,Nom6], Rem) :-
    [ifeil(L1,[Typ,|],Nom1,Lsf xl)],
    fael_adwt(Lsf xl,L2,Nom2,Jhmn2,L3,Lsf x2),
    mfol_adwt(Lsf x2,L3,Nom3,Jhmn3,L4,Lsf x3),
    mfol_adwt(Lsf x3,L4,Nom4,Jhmn4,L5,Lsf x4),
    mfol_adwt(Lsf x4,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], [Nom1,Nom2,Nom3], Rem) :-
    [ifeil(L1,[Typ,|],Nom1,Lsf xl)],
    naeb_faei_adwt(Lsf xl,L2,Nom2,Jhmn2,L3,|$$|),
    tbi_fli_adwt(L3,Nom3,Jhmn3,Rem),
    [X,Nom]=Nom2, asma_marfo(X,Nom).
```
afb([L1;L2],[Nom1,Nom2,Nom3,Nom4],Rem)::
  [feil(L1,T,y$_{2}$),Nom1,Lsf1)!!,
   naeb_fael_adwt(Lsf1,L2,Nom2,Jhmn2,L3,Lsf2),
   mfol_adwt(Lsf2,L3,Nom3,Jhmn3,L4,[$$]),
   tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
   [X,Nom]=Nom2, asma_marfo(X,Nom),

afb([L1;L2],[Nom1,Nom2,Nom3,Nom4,Nom5],Rem)::
  [feil(L1,T,y$_{3}$),Nom1,Lsf1)!!,
   naeb_fael_adwt(Lsf1,L2,Nom2,Jhmn2,L3,Lsf2),
   mfol_adwt(Lsf2,L3,Nom3,Jhmn3,L4,Lsf3),
   mfol_adwt(Lsf3,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,Nom7]=Nom4, asma_mnsb(M2,Nom7).

afb(L,$[$],Rem)::
  ma_tfdeel(L,Nom,Rem).

afb(L,$[$],Rem)::
  ma_tfdeel(L,Nom,Rem).

afb(L,$[$],Rem)::
  ma_tfdeel(L,Nom,Rem).

ma_tfdeel([L1,L2:L3],[Nom1,Nom2,Nom3,Nom4],Rem)::
  ifthen((var(L2)),(esm_marfo($mety$_{1}$),Nom1,_,
    feil_am($ty$_{3}$),Nom1,_,
    Nom2=[N,Nom1]), ma(L1,Nom1),
    feil(L2,$ty$_{3}$),Nom3,[$$],Nom2,Lsf2),
    mfol_adwt(Lsf2,L3,Nom3,Jhmn3,L4,_,
    tbi_fli_adwt(L4,Nom4,Jhmn4,Rem),
    esm_marfo($mety$_{1}$),Nom1,_,
    [N,Nom]=Nom2, feil_am($ty$_{3}$),Nom1,_,
    irab_jmlh($ty$_{3}$),
    {N,Nom2}={N,Nom2},
    irab_jmlh($ty$_{3}$),
    ma(L1,Nom1), feil(L2,$ty$_{3}$),Nom2,_,
    an(L3,Nom3), afb1(L4,Nom4,Rem),
    esm_marfo($mety$_{1}$),Nom1,_,
    [N,Nom]=Nom2,
    f_mndi(N,Nom,_,
    irab_jmlh($ty$_{3}$),
    {N,Nom2}={N,Nom2},
    irab_jmlh($ty$_{3}$),
    adat(Nom3,_,
    [N4,Nom44]=Nom4, f_mansob(N4,Nom44,_,
    irab_jmlh($ty$_{3}$),
    $ty$_{3}$, [N,Nom]=Nom2, f_mansob(N4,Nom44,_,
    irab_jmlh($ty$_{3}$).
afb2_an([L1:L2],[[$خبر$,Nom1,Nom2],[[$خبر$,Nom3],Rem]) :-
  an(L1,Nom1),
  ifthen( (var(L2)),([$_,[E1,E2:_]=Nom2,
  [Fl,F2]=E1, f_mansob(Fl,F2,$_), E1=[Fl,F2],
  Nom2=[$_,[E1,E2:_]=Nom2,
  asma_marfo($خبر$,Nom3,$_))]),
  bfb(L2,Nom2,L3),khbr_dw1(L3,Nom3,Jhm3,Rem),
  adat(Nom1,_),
  [$_,[E1,E2:_]=Nom2,
  [Fl,F2]=E1, f_mansob(Fl,F2,$_),
  irab_jmlh($خبر$),
  asma_marfo($خبر$,Nom3,$_).

afb2_an([L1:L2],[[$خبر$,Nom1,Nom2],[[$خبر$,Nom3],Rem]) :-
  an(L1,Nom1), afbl(L2,Nom2,L3),
  khbr_dw1(L3,Nom3,Jhm3,Rem), adat(Nom1,_),
  [$_,[E1,E2:_]=Nom2, [Fl,F2]=E1,
  f_mansob(Fl,F2,$_),
  irab_jmlh($خبر$),
  asma_marfo($خبر$,Nom3,$_).

afb2_an([L1:L2:L3],[[$خبر$,Nom1,Nom2,Nom3],[[$خبر$,Nom4],Rem]) :-
  an(L1,Nom1), la(L2,Nom2),
  afbl(L3,Nom2,L4), [[$_$_]=Nom2,
  khbr_dw1(L4,Nom4,Jhm3,Rem),
  adat(Nom1,_),
  adat(Nom2,_),
  [N2,Nom2]=Nom2,
  f_mansob(N2,Nom2,$_),
  irab_jmlh($خبر$),
  asma_marfo($خبر$,Nom4,$_).

afb2_estfhm([L1:L2],[Ninfo,Nom],Rem) :-
  hl(L1,Ninfo), adat(Ninfo,$_),
  ((bfb(L2,Nom,Rem),[$_,[Fl,F2:_]=Nom,feil_am(Fl,F2,$_));
  (afbl(L2,Nom,Rem),[$_,[Fl,F2:_]=Nom,feil_am(Fl,F2,$_));
  (afb2_feil(L2,Nom,Rem),[$_$_]=Nom,
  irab_jmlh($خبر$),
  asma_marfo($خبر$)).

afb2_estfhm([L1:L3],[Ninfo,Nom],Rem) :-
  hamza(L1,Ninfo,L2),
  adat(Ninfo,$_),
  ((bfb(L2,Nom,Rem),[$_,[Fl,F2:_]=Nom,feil_am(Fl,F2,$_));
  (afbl(L2,Nom,Rem),[$_,[Fl,F2:_]=Nom,feil_am(Fl,F2,$_));
  (afb2_feil(L2,Nom,Rem),[$_$_]=Nom,
  irab_jmlh($خبر$),
  asma_marfo($خبر$)).
 afl2_estfhm([L1;L2],[Ninfo,Nom],Rem) :-
  esm_estfhm($\text{ feliz}$,L1,Ninfo),
  esm_marfo($\text{ مريحة}$,Ninfo,_,
  ((bfb(L2,Nom,Rem),[X,[[F1,F2]:_]]\text{ feliz},fei_am(F1,F2,_))
  (afbl(L2,Nom,Rem),[X,[[F1,F2]:_]]\text{ feliz},fei_am(F1,F2,_))
  afl2_feil(L2,Nom,Rem),[X:_]\text{ feliz}.
  irab_jmlh($\text{ رقم}$,X).

 afl2_qasam(L,[Nom1,Nom2],Rem) :-
  afbl(L2,Nom,Rem),afbl(L2,Nom,Rem),
  irab_jmlh($\text{ جملة}$,$\text{ رفع}$).

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

 fael_adwt(Lsfx,L,[$\text{ feliz}$,Nom],Jhmn,Rem,Rsfx) :-
  ifthen( (var(L),
  asma_marfo($\text{ feliz}$,Nom)),
  khbr_dwt3(L,Nom,Jhmn,Rem), asma_marfo($\text{ feliz}$,Nom).

 mfol_adwt(Lsfx,L,[$\text{ feliz}$,Nom],Jhmn,Rem,Rsfx) :
  khbr_dwtl(L,Nom,Jhmn,Rem).

 tbi_fli_adwt(L,[Nom1,Nom2],Jhmn,Rem) :-
  ifthenelse( (var(Nom1),var(L)),
  (tbi_fli(L2,Nom2,Jhmn,Rem), (jr_mjro(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( (\text{ feliz} var(L)),(length(L,N))),
  art_test(L,[Adat,Rt,De,Typ,\text{ feliz},Sfx],N),
  member(Adat,[\text{ feliz} $\text{ feliz}$,\text{ feliz},\text{ feliz},\text{ feliz},\text{ feliz},\text{ feliz}]),
  Ninfo=[Adat,Rt,De,Typ,\text{ feliz},Sfx].

 ma(L,Ninfo) :-
  ifthen( (\text{ feliz} var(L)),(length(L,N))),
  art_test(L,Ninfo,N), xtrct(L,Ninfo,Adat), Adat=$\text{ feliz}$.

 la(L,Ninfo) :-
  ifthen( (\text{ feliz} var(L)),(length(L,N))),
  art_test(L,Ninfo,N), xtrct(L,Ninfo,Adat), Adat=$\text{ feliz}$.
bam(L, [Nom], [Pred], Rem) :-
    mbtda_mrk(L, Nom, [Jns], Hall, _, Rem),
    khbr_mrk(Rem, Pred, [Jns2], Hall2, _, Rem),
    mtcjhs([Jns1], Hall1, [Jns2], Hall2, [Jns], Hall).

bam(L, [Nom], [Pred], Rem) :-
    mbtda_mrk(L, Nom, [Jns1], Hall1, _, Rem),
    khbr_mrk(Rem, Pred, [Jns2], Hall2, _, Rem),
    mtcjhs([Jns1], Hall1, [Jns2], Hall2, [Jns], Hall).

bam(L, [Nom], [Pred], Rem) :-
    mbtda_mrk(Rem, Pred, [Jns2], Hall2, _, Rem),
    khbr_mrk(Rem, Pred, [Jns2], Hall2, _, Rem),
    mtcjhs([Jns1], Hall1, [Jns2], Hall2, [Jns], Hall).

mbtda_mrk([L1|L2], [Nom], [Nom1, Nom2], Jhmn, Rem) :-
    mowsol_esm(L, Nom, Jhmn1),
    khbr_jml(L2, Nom2, Rem),
    xect(L, Nom2, X), ifthen((X=\$P\$, X)),
    (xect(Z, Nom2, [[Y], Z, _, _]), fail_am(Y, Z, _)),
    irab_jmlh($L\$, X).

mbtda_mrk(L, [Nom], [Nom1, Nom2, Nom3], Jhmn, Rem) :-
    mbtda(L, Nom1, Jhmn1, [L2|L3]),
    esm_mowsol(L2, Nom2, Jhmn2),
    khbr_jml(L3, Nom3, Rem),
    xect(L, Nom3, X), ifthen((X=\$P\$, X)),
    (xect(Z, Nom3, [[Y], Z, _, _]), fail_am(Y, Z, _)),
    irab_jmlh($L\$, X).

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

khbr_mrk(L, Nom, Jhmn, Rem) :-
    khbr(L, Nom, Jhmn, Rem).

khbr_mrk(L, Nom, Jhmn, Rem) :-
    mbtda_mrk(L, Nom, Jhmn).

khbr_mrk(L, Nom1, Nom2, Jhmn, Rem) :-
    esm_mnkra_mkrr(L, Nom1, Jhmn1, L1),
    khbr_jml(L1, [\$p\$, 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),
            asma_marfo(Y1, Y2), asma_marfo(Z1, Z2).

mowsol_esm(L, [\$s\$, Nom], [Jns, Hal, \$s\$, \$m\$]) :-
    ifthen((\+, var(L)), (length(L, N)), art_test(L, Nom, N),
           xect(L, Nom, Wrd),
           member(Wrd, [\$l\$, \$d\$, \$h\$, \$l\$, \$d\$, \$h\$, \$l\$, \$d\$, \$h\$, \$l\$, \$d\$, \$h\$, \$l\$, \$d\$]),
           bal_jns(Wrd, Jns, Hal))!
bfm(L,[Nom1,Nom2],Rem) :-
  ifthen( (var(L)),( xtrct(2,Nom1,[_,-,Famr]:_),
    [_,-,_,_=Famr,
    xtrct(2,Nom2,[[_,_,_,_=Famr]:_]:_)),
    xtrct(2,Nom2,[[_,_,_,=Famr]:_]:_),
    xtrct(1,Nom2,X),
    feil_am($,Famr,_),
    feil_am($,فپیا نیمیزی نیمیزی، فقط $،%,-,_)),
    xtrct(2,Nom1,[[_,_,Famr]:_]),
    xtrct(2,Nom2,[[_,_,_,=Famr]:_]:_),
    bfb(L,Nom1,L1),
    bfb(L,Nom2,Rem),xtrct(2,Nom2,[[_,_,_,=Famr]:_]:_),
    xtrct(1,Nom2,X),
    feil_im($،%,-,Famr,_),
    feil_am($،%,-،$،Z,_),
    irab_jmlh($،%,-،$،X).

bfm([L1:L2],[Nom1,Nom2],Rem) :-
  feil(L1,_,Nom1,LsfX),
  ifthenelse((LsfX=[$]),(L3=L2),((L3 = [LsfX:L2])),
  bam(L3,Nom2,Rem),xtrct(2,Nom2,[_,-,Fael]),
  asma_marfo($،%,-,$،Y),
  xtrct(3,Nom2,[_,-,Hal]),
  asma_mnsb($،%,-,$،Hal),
  [Jml_Typ,[_,-,[_,-,Typ:_]:_]:_]=Hal,
  ifthen((Jml_Typ = $،%,-،$،Typ:$_),
  irab_jmlh($،%,-،$،Typ)).

bfm([L1:L2],[Nom1,Nom2],Rem) :-
  ifthen( (var(L1)),
    [_,-,[Y,-,[Z]=Nom2,asma_marfo($،%,-,$،Y),
    xtrct(3,Nom2,[_,-,Z]), asma_mnsb($،%,-,$،Z),
    Nom2=[_,-,[Y,-,[Z]]] ),
    bam(L2,Nom2,Rem),
    [_,-,[Y,-,[Z]=Nom2,asma_marfo($،%,-,$،Y),
    xtrct(3,Nom2,[_,-,Z]), asma_mnsb($،%,-,$،Z),
    Nom2=[_,-,[Y,-,[Z]]].

bfm([L1:L2],[Nom1,Nom2],Rem) :-
  ifthen( (var(L1)),
    kan(L1,Nom1,Jhl), xtrct(2,Nom1,[_,-,_,_,_,Sfx]),
    ifthen((Sfx=[$,len(L1,3)),
    (Nom2=[_,-,[Sfx:$_]]),
    esm_kan(Sfx,Nom2,[Jns,Hal,$،%,-,$，مغزه$]),
    khbr_mrkb(L2,Nom3,[Jns2,Hal2,_,Rem],
    asma_marfo($،%,-,$،Y),
    xtrct(3,Nom2,[_,-,Z]), asma_mnsb($،%,-,$،Y),
    Nom2=[_,-,[Y,-,[Z]]].

bfm([L1:L2],[Nom1,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($،%,-,$،Y),
  irab_jmlh($،%,-،$،
  جملة من الفعل والفاعل $،
  xtrct(2,Pred,Nom3),
  asma_marfo($،%,-,$،Y),
  Nom3=[_,-,[Y,-,[Z]]].

الجملة في معدل رقم خبر $, Z$

khbr_jml_dwt(L, Nom, Rem) :- afb(L, Nom, Rem),
            xtract(1, Nom, X),
            jml_typ(X, Y),
            ifthen( (Y==$، $)،
                    (xtract(2, Nom, [[Y, Z|: _]], feil_am(Y, Z, _)))).
afm(L,[L:1:L5],Nom1,Nom2,Nom3,Rem) :-
  f(L,Nom1,[L:1:L5]), hrf_atf(L2,Nom2,L4),
  ifthenelse((L4==[$$]),(L3=L5),(L3=[L4:L5]),
  fl(L3,Nom3,Rem).

afm(L,[L:1:L5],Nom1,Nom2,Nom3,Rem) :-
  f(L,Nom1,[L:1: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).

fl(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,$$$}),
  art_test(L,Nom,N),xtrct(l,Nom,Adat),
  member(Adat,[$$|$,$$$|$,$$$|$,$$$|$,$$$|$,$$$|$,$$$|$,$$$|,$$$|$,$$$|,$$$}),
  $,$,$,$,$,$,$,$,$,$,$.$

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

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

afm_estfhm([L1:L2],Ninfo,Nom,Rem) :-
  esm_estfhm($$$),L1,Ninfo), esm_mnsb($$$),Ninfo, ),
  f_est(L2,Nom,Rem).
/*----------------------------------------*/
/*----------------------------------------*/

NOUN END-CASE SUB-MODULE

/*----------------------------------------*/
/*----------------------------------------*/

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
esm_marfo(X0,[X1,X2,_,_,_,_,_,_,_,_,Part], Out ) :=-
  member(X2,[X0,$,X1,X2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
اسم مرجع%

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).

$\text{esm.marfo}(X0, [X1, _, X3, _, _], Out) :-$
  member( X1, [X0, _]),
  Out = $\text{esm.marfo}(X1, [X0, _], Out)$,
  rite([Out, 'X0', 'X1'], nl).
البيانات في الجدول السابق:

<table>
<thead>
<tr>
<th>عنصر X1</th>
<th>عنصر X2</th>
<th>عنصر X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

البيانات في الجدول التالي:

<table>
<thead>
<tr>
<th>عنصر X1</th>
<th>عنصر X2</th>
<th>عنصر X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
</tr>
</tbody>
</table>

البيانات في الجدول التالي:

<table>
<thead>
<tr>
<th>عنصر X1</th>
<th>عنصر X2</th>
<th>عنصر X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$y_2$</td>
<td>$y_3$</td>
</tr>
</tbody>
</table>

البيانات في الجدول التالي:

<table>
<thead>
<tr>
<th>عنصر X1</th>
<th>عنصر X2</th>
<th>عنصر X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>$z_2$</td>
<td>$z_3$</td>
</tr>
</tbody>
</table>

البيانات في الجدول التالي:

<table>
<thead>
<tr>
<th>عنصر X1</th>
<th>عنصر X2</th>
<th>عنصر X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
</tr>
</tbody>
</table>
VERB END-CASE SUBMODULE

f_madi(X0, [X1, X2, X3, $α$], Out ) :
  member(Sfx, [$$, $α$, $β$, $γ$, $δ$, $ε$, $θ$, $ρ$, $ς$, $ζ$, $μ$, $κ$, $λ$, $ν$, $ξ$, $ο$, $π$, $τ$, $φ$, $ψ$, $φ$, $ς$]), 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===$α$; H===$β$),
  Out = $α$ ماضٍ مبنيً على الفتحة المقدسة للحذف $α$,
  rite([Out,X0,':',X,':',X1 ] ), nl.

f_madi(X0, [X1, X2, X3, $α$], Out ) :
  member(Sfx, [$$, $α$, $β$, $γ$, $δ$, $ε$, $θ$, $ρ$, $ς$, $ζ$, $μ$, $κ$, $λ$, $ν$, $ξ$, $ο$, $π$, $τ$, $φ$, $ψ$, $φ$, $ς$]), string_length(Sfx, Lnth2 ),
  ifthen( ( \+ var(X1) ), ( string_length(X1, Lnth1 ),
   Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X))),
  Out = $α$ ماضٍ مبنيً على الفتحة المقدسة $α$.

f_madi(X0, [X1, X2, X3, $α$], Out ) :
  ifthen( ( \+ var(X1) ), ( string_length(Sfx, Lnth2 ),
   string_length(X1, Lnth1 ), Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X))),
  Out = $α$ ماضٍ مبنيً على السكون لتصاغة تضيير رفع متحرك $α$,
  rite([Out,X0,':',X,':',X1 ] ), nl.

f_amr(X0, [X1, X2, X3, $α$], Out ) :
  substring(X2,0,1,H),
  member(H, [$$, $α$, $β$, $γ$, $δ$, $ε$, $θ$, $ρ$, $ς$, $ζ$, $μ$, $κ$, $λ$, $ν$, $ξ$, $ο$, $π$, $τ$, $φ$, $ψ$, $φ$, $ς$]),
  ifthen( ( \+ var(X1) ), ( string_length(Sfx, Lnth2 ),
   string_length(X1, Lnth1 ), Lnth is Lnth1 - Lnth2,
   substring(X1,Lnth2,Lnth,X))),
  Out = $α$ أمر مبنيً على مضاف حرف العلة $α$,
  rite([Out,X0,':',X,':',X1 ] ), nl.

f_amr(X0, [X1, X2, X3, $α$], Out ) :
  member(Sfx, [$$, $α$, $β$, $γ$, $δ$, $ε$, $θ$, $ρ$, $ς$, $ζ$, $μ$, $κ$, $λ$, $ν$, $ξ$, $ο$, $π$, $τ$, $φ$, $ψ$, $φ$, $ς$]), string_length(Sfx, Lnth2 ),
  ifthen( ( \+ var(X1) ), ( string_length(X1, Lnth1 ),
   Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X))),
  Out = $α$ أمر مبنيً على السكون $α$,
  rite([Out,X0,':',X,':',X1 ] ), nl.
الأمر يبني على السؤال عبر النص

f_amr(XO,[X1,X2,X3,$م,$_,$_,$Sfx,$_,$_,$],Out):=
    ifthen( (++ var(X1)) ,
        (string_length(Sfx, Lnth2 ), Lnth2 >=1,
        B is Lnth2-1,
        substring(Sfx,B,1,H),
        H=$((,$م,$_,$_,$Sfx,$_,$_,$),
        string_length(X1, Lnth1 ),
        Lnth is Lnth1 - Lnth2,
        substring(X1,Lnth2,Lnth,X)),
        Out = $ ميني على السؤال,$,
        rite([Out,X0,'=''X,'=''X1]), nl.

f_majzom(X0,[X1,X2,X3,$م,$_,$_,$Sfx,$_,$_,$Jns,Hal,_],Out):=
    ( member(Sfx,[|$,$,$|$,$,$|$,$]),
    (substring(Sfx,Lnth2-1,1,H),member(II,[|$,$,$|$,$,$|$,$]),
    string_length(Sfx, Lnth2 ), Lnth2 >=1,
    ifthen( (++ var(X1)) ,
    (string_length(X1, Lnth1 ),
    Lnth is Lnth1 - Lnth2, substring(X1,Lnth2,Lnth,X)),
    Out = $ ممارة ميزوم علی التّج للفعل الفعلة,$,
    rite([Out,X0,'=''X,'=''X1]), nl.

f_majzom(X0,[X1,X2,X3,$م,$_,$_,$Sfx,$_,$_,$Jns,Hal,_],Out):=
    substring(X2,0,1,H),
    member(II,[|$,$,$|$,$,$|$,$,$|$,$,$|$,$]),
    ifthen( (++ var(X1)) ,
    (string_length(Sfx, Lnth2 ),string_length(X1, Lnth1 ),
    Lnth is Lnth1 - Lnth2,substring(X1,Lnth2,Lnth,X)),
    Out = $ ممارة ميزوم علی التّج للفعل الفعلة,$,
    rite([Out,X0,'=''X,'=''X1]), nl.

f_mansob(X0,[X1,X2,X3,$م,$_,$_,$Sfx,$_,$_,$Jns,Hal,_],Out):=
    ( member(Sfx,[|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,$,$|$,
f_marfo(X0,[X1,X2,X3,$$,$$,$$,$$,Sfx,$$,X1],Out):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).

f_marfo(X0,[X1,X2,X3,$$,$$,Sfx,$$,Out]):
    Out = ممارع مرفوع بالحجة على آخرة $,
    rite([Out,X0,':',X1,X1], nl).
/*--------------------------------------------------------*/
/*  */
/*  */
/*  */
/*--------------------------------------------------------*/

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

find_art2([A,B],2,$h$,$d$,$e$,$g$,$j$,$v$,$B$,,$S$,,$h$) :-
  member(Root,[sh$,gd$]),
  conca([A,B],Root).

find_art2([A,B],2,Root,$d$,$h$,$u$,$e$,,$e$,,$S$,,$G$,,$G$,,$h$,,$S$,,$h$) :-
  member(Root,[jG$,Ge$,$fe$]),
  conca([A,B],Root).

find_art2([A,B],2,Root,$j$,$a$,$G$,,$a$,$Q$,,$M$,,$S$,,$h$,$S$,,$h$) :-
  member(Root,[Gd$,Ge$]),
  conca([A,B],Root).

find_art2([A,B],2,Root,$Q$,,$O$,,$U$,e,$S$,,$G$,,$M$,,$S$,,$h$,$S$,,$h$) :-
  member(Root,[fC$,Ge$,$j$,$e$,$t$,$C$,,$sh$]),
  conca([A,B],Root).

find_art3([A,B,C],2,Root,$I$,O,T,G,$e$,$G$,,$e$,,$S$,,$C$,,$G$,,$h$,,$S$,,$h$) :-
  member(C,[h$,d$,$c$]),
  member(Root,[GP$,iJ$,gJ$,jP$,jP$,]),
  conca([A,B],Root).

find_art3([A,B,C],2,Root,$d$,$U$,J,$e$,,$Q$,,$j$,$e$,$v$,,$C$,,$G$,,$h$,,$S$,,$h$) :-
  member(C,[h$,d$]),
  member(Root,[Gg$,eg$,$fg$]),
  conca([A,B],Root).

find_art3([A,B,C],2,Root,$A$,G,f,$K$,J,$S$,G,a,$Q$,,$M$,,$S$,,$h$,$S$,,$h$) :-
  Root=GdG$,
  conca([A,B,C],Root).

find_art4([A,B,C,D],2,Root,$d$,$h$,$u$,$e$,$e$,$G$,,$S$,,$h$,$S$,,$h$) :-
  member(Root,[jPdG$,iJdG$,GPe$]),
  conca([A,B,C,D],Root).

find_art6([A,B,C,D,E,F],2,Root,$d$,$h$,$u$,$e$,$S$,,$G$,,$S$,,$h$,$S$,,$h$) :-
  member(F,[h$,d$,$c$]),
  member(Root,[fjPdG$,jCdG$]),
  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], [A,B], [Oword, $الف, $مفرد, $$,$$$,$$$,$$$]) :-
    conca([A,B], Oword).

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

noun_test3([A,B,C], [Oword, Oword, DE, $الف, $مفرد, $$,$$$,$$$,$$$]) :-
    member(C, [A,B,C], [A,B,C]),
    conca([A,B,C], Oword),
    concat([A,B,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),
    conca([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),
    conca([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),
    conca([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),
    conca([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),
    conca([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),
    conca([A,B,C,D,E,F], Oword).
noun_test6([A, B, C, D, E, F], [Oword, RO, DR, SDP, MF, PR, IN, SUJ]) :-
npre_test6([A, B, C, D, E, F], RO, DR, 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, SUJ]) :-
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, SUJ]) :-
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, DR, SDP, MF, PR, IN, SU]) :-
npre_test8([A, B, C, D, E, F, G, H], RO, DR, 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, DK, SDP, MF, PR, IN, SU]) :-
nsuf_test8([A, B, C, D, E, F, G, H], RO, DK, 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_test11([A, B, C, D, E, F, G, H, I, J, K],
[Oword, RO, DE, SDP, MF, PR, IN, SU]) :-
npre_test11([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_test12([A, B, C, D, E, F, G, H, I, J, K, L],
[Oword, RO, DE, SDP, MF, PR, IN, SU]) :-
npre_test12([A, B, C, D, E, F, G, H, I, J, K, L], RO, DE, SDP, MF, PR, IN, SU),

noun_test13([A, B, C, D, E, F, G, H, I, J, K, L, M],
[Oword, RO, DE, SDP, MF, PR, IN, SU]) :-
pre_test13([A, B, C, D, E, F, G, H, I, J, K, L, M], RO, DE, SDP, MF, PR, IN, SU),

npre_test4([A, B, C, D], RO, DR, $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$)،
member(PR, [$\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$， $\mu$، $\mu$، $\mu$، $\mu$، $\mu$، $\mu$)،
conca(C, D, PR),
DE = $\mu$،
conca([A, B], RO).
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,12) :- verb_test12(L,T).

verb_test2([A,B], [Oword,R,$3$,$2$,$1$,$X,Y,Z,$2$,$3$]) :-
  member(A,[A,B]),
  concat([A,B],Oword).

verb_test3([A,B,C], [Oword,Oword,$3$,$2$,$1$,$X,Y,Z,$2$,$3$]) :-
  member(Oword,[Oword,$3$,$2$,$1$]),
  concat([A,B,C],Oword).

verb_test3([A,B,C], [Oword,R,$3$,$2$,$1$,$X,Y,Z,$2$,$3$], Sfx) :-
  member(C,[A,B,C]),
  member(Sfx,[Sfx,$3$,$2$,$1$]),
  concat([A,B,Sfx], concat([Sfx,C,$3$],R)),
  concat([Sfx,C],Oword).

verb_test4([A,B,C,D], [Oword,RO,DE,Ty,X,Y,Z,$2$,$3$]) :-
  vinf_test4([A,B,C,D], [RO,DE,Ty,X,Y,Z,$2$]),
  concat([A,B,C,D],Oword).

verb_test4([A,B,C,D], [Oword,RO,DE,Ty,X,Y,Z,$2$,$3$], SU) :-
  vsuf_test4([A,B,C,D], [RO,DE,Ty,X,Y,Z,$2$], SU),
  concat([A,B,C,D],Oword).

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

verb_test6([A,B,C,D,E,F],
  [Oword,Oword,$3$,$2$,$1$,$X,Y,Z,$2$,$3$]) :-
  concat([A,B,C,D,E,F],Oword), Oword=[A,B,C,D,E,F].

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

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

verb_test10([A, B, C, D, E, F, G, H, I, L, J], 
[Oword, RO, DE, Ty, X, Y, Z, PR, IN, SU]) :- 
vsuf_test10([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_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, J, K], 
[Oword, RO, DE, Ty, X, Y, Z, PR, IN, SU]) :- 
vpre_test12([A, B, C, D, E, F, G, H, I, J, K], RO, DE, Ty, X, Y, Z, PR, IN, SU), 
concat([A, B, C, D, E, F, G, H, I, J, K], Oword).

vpre_test4([A, B, C, D], RO, DE, TY, $\text{مفرد،_}{}$,$\text{بـ،}$,$\text{D,}$,$\text{SS}$) :- 
member(D, [$\text{D}$,$\text{SS}$]), 
conca([A, B, C], RO), 
DE=$\text{امضي}$, 
TY=$\text{فعل}$.

vpre_test5([A, B,$\text{بـ،}$,$\text{E,}$], RO, DR, $\text{مفرد،}$,$\text{مذكر،}$,$\text{تَمْرِيحة،}$,$\text{R,}$,$\text{SS}$) :-member(E, [$\text{E}$]), 
DE=$\text{فعل}$, 
conca([A, B, D], RO).

vpre_test5([A, B, C, D, E], RO, DR, TY, $\text{مفرد،}$,$\text{مذكر،}$,$\text{PR,}$,$\text{SS}$) :-member(D, [$\text{D}$]), member(E, [$\text{E}$]), conca([A, B, C], RO), 
concat([D, E], PR), TY=$\text{فعل}$, DR =$\text{فعل}$.

vsuf_test5([A, B, C,$\text{بـ،}$,$\text{E,}$], RO, DE, $\text{مذكر،}$,$\text{X,}$,$\text{Y,}$,$\text{Z,}$,$\text{SS}$, SU) :- 
conca(A, B, SU), member(SU, [$\text{SU}$]), member(SU, [$\text{SU}$]), 
DE=$\text{فعل}$, 
vsuf_test10([$\text{I}$,$\text{K}$,$\text{L}$,$\text{M}$,$\text{N}$,$\text{O}$,$\text{P}$,$\text{Q}$,$\text{R}$,$\text{SS}$], RO, DR, TY, $\text{تموزيه}$,$\text{مذكر،}$,$\text{SS}$) :- DR =$\text{فعل}$, 
conca([H, I, L, J], RO), TY=$\text{فعل}$.

vsuf_test11([$\text{I}$,$\text{K}$,$\text{L}$,$\text{M}$,$\text{N}$,$\text{O}$,$\text{P}$,$\text{Q}$,$\text{R}$,$\text{SS}$], RO, DR, TY, $\text{تموزيه}$,$\text{مذكر،}$,$\text{SS}$) :- DE =$\text{فعل}$, 
conca([H, I, L, J], RO), TY=$\text{فعل}$.
REFERENCES


