Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

An Intelligent Framework for Natural Language Stems Processing

Abid Thyab Al Ajeeli

Received: 14 December 2015 Accepted: 5 January 2016 Published: 15 January 2016

Abstract

This work describes an intelligent framework that enables the derivation of stems from inflected words. Word stemming is one of the most important factors affecting the performance of many language applications including parsing, syntactic analysis, speech recognition, retrieval systems, medical systems, tutoring systems, biological systems,?, and translation systems. Computational stemming is essential for dealing with some natural language processing such as Arabic Language, since Arabic is a highly inflected language. Computational stemming is an urgent necessity for dealing with Arabic natural language processing. The framework is based on logic programming that creates a program to enabling the computer to reason logically.This framework provides information on semantics of words and resolves ambiguity. It determines the position of each addition or bound morpheme and identifies whether the inflected word is a subject, object, or something else. Position identification (expression) is vital for enhancing understandability mechanisms. The proposed framework adapts bi-directional approaches. It can deduce morphemes from inflected words or it can build inflected words from stems. The proposed framework handles multi-word expressions and identification of names.

Index terms— natural language, knowledge base, morphological analysis, inflected words, logic-based, definite-clause, context-free grammar.

1 I. Introduction

ord stemming is one of the most important factors affecting the performance of many natural language processing applications such information retrieving systems, machine pattern recognition, machine translation, speech tagging, and many other systems. The study of natural language processing using computers is as old as the introduction of computer software. Advances in this area lead to the improvement of man-machine communications. In particular, interfaces to databases and intelligent systems can handle interaction in restricted subsets of natural languages.

Natural language processing is a computer activity in which computers are entailed to analyze, understand, alter, or generate natural language objects. This includes the automation of any or all linguistic forms, activities, or methods of communication, such as conversation, correspondence, reading, dictation, publishing, translation, lip reading, ?, and written composition. Natural language processing is also the name of the branch of computer science, artificial intelligence, and linguistics concerned with enabling computers to engage in communication using natural language(s) in all forms, including but not limited to speech, print, writing, and signing (Foss 2004).

The Arabic language is one of the major natural languages that place a great deal of emphasis on morphology and syntax. More than 250 million people speak it. Arabic is a synthetic or in other words, highly inflected language. This means that the syntactic relationship between nouns are indicated by case ending and that verbs are inflected by means of prefixes, infixes, and suffixes to indicate the various persons, numbers, genders, derived forms, moods, and tenses. Traditional Arabic morphology is described in terms of roots and patterns. Roots are not words but sequences of three or more consonants.

In addition, Arabic language has a number of implications for the design of computer systems in general. It has also quite distinguishable characteristics that will add new aspects to natural language processing. The

technique used in this work is based on two-level descriptions of Arabic morphology. This approach will make affixes (prefixes, infixes, and suffixes) more easily understood and handled. The twolevel descriptions will be based on finite-automata and regular expressions. But, first, a review of the basic concepts of finite automata and the basic text processing machines will be introduced and adapted to capture Arabic structure concepts (Kelley, 1995;Hopcroft & Ullman 1979).

Formally, a finite automaton is defined by a 5tuple (Q, ?, ?, q 0 , F) where:

? Q is a finite set of states q 0 , q 1 , ..., q n, ? ? is a finite input alphabet (a set of symbols), ? q 0 ? Q is the initial state,

? F ? Q is the set of final sates, and ? ? is the partial transition function mapping Q × (? + {?}) to zero or more elements of Q.

For example, ?(q, ???) describes the next states, for each state q and input symbol ??? or it is undefined. The notion of using finite automata is to model lexical analyzer, syntactic analyzer, and even to associate semantics with morphemes.

In this paper, we describe a software system that has the capability of embodying some parts of expert's facilities. The grammar rules of Arabic language are transformed into first-order predicate logic, using the programming language Prolog. The rules are developed to analyze and to extract semantic information from Arabic texts. The first-order predicate logic provides three levels of analysis:

1. Syntax: Which deals with grammatical structure of the underlying natural language. In addition to, the opportunity of encoding meaning.

2. Semantics: Deals with the literal meaning with the opportunity of including rules for wider meaning contexts, and 3. Pragmatics: Deals with the real meaning of sentences.

Stemming is a heuristic process that chops off the ends of words to find the root word and often includes the removal of derived affixes. It is used to improve retrieval effectiveness and to reduce the size of indexing files. Stemming is a common method for morphological normalization of natural language texts. Modern information retrieval systems rely on such normalization techniques for automatic document processing tasks. High quality stemming is sometimes difficult in highly inflectional languages, for example, Arabic and Indic languages. Little research has been performed on designing algorithms for stemming of texts in those languages **??**Sahari N. et al., 2013).

Computational stemming is an urgent problem for Arabic Natural Language Processing, because Arabic is a highly inflected language. The existing stemmers have ignored the handling of multi-word expressions and identification of Arabic names **??**Alhanini & Abo Aziz 2011). For other inflected languages. Jain and Agrawa (2015) claimed that they manage to parse, for example, Hindi words to identify root words from inflected words using natural language processing (NLP) techniques.

In order to familiarize readers with the complexities involved in the analysis and construction of Arabic sentences, we illustrate briefly some of the funda-mental features of Arabic (Ali 1988; Hamoody 1991; Al-Douri 1992; Al Daimi 1994).

? Arabic script is written from right to left.

? Arabic language is an inflectional language and the derivation in Arabic is based on morphological patterns and the verb plays a greater inflectional role. Furthermore, Arabic words are built up from roots representing lexical and semantic connecting elements. This is not the case, for example, with English, which employs the stem as a basis for word generation. ? Arabic offers the possibility of combining particles and affixed pronouns to words and it involves diacritization. ? Arabic is distinguished by its high syntactical flexibility. This flexibility includes: the omission of some prepositional phrases associated with verbs; the possibility of using several prepositions with the same verb while preserving the meaning; allowing more than one matching case between the verb and verbal subject and the adjective and its broken plural qualified and the sharpness of pronominalization phenomena where the pronouns usually indicate the original positions of the words before their extra-positioning, fronting and omission.

In other words, Arabic allows a great deal of freedom in the ordering of words in a sentence. Thus, the syntax of the sentence can vary according to transformational mechanisms such as extraposition, fronting and omission, or according to syntactic replacement such as the use of an agent noun in place of a verb. ? Arabic language is distinguished by its high context sensitivity in several directions. On the writing level, the shape of a letter depends on the letter that precedes it and the one that follows it. On the syntactic level, the different synthetic coherence relations such as case ending, matching, connecting, associating and pro-nominalization represent various examples of syntactic context sensitivity. Furthermore, the context sensitivity feature extends to the lexicon where a lot of vocables are influenced by their associated words.

The context sensitivity feature is not only limited to letters, words and sentences but also applied to the continuous context consisting of several sentences. Arabic sentences are embedded and normally connected by copulative, exceptive and adversative particles. For this reason it is more difficult to identify the end of an Arabic sentence than is the case with other languages.

There are a number of applications that directly borrow models and methods from both information retrieval (IR) and natural language processing (NLP). A short presentation of some of these applications is mentioned below **??**Indurkhya et This paper is organized into a number of sections. The next section provides some background on related works. Section 3 provides a brief description of lexical analysis mechanisms. It points out

the relationship between lexical analysis processes and finite automata. Section 4 introduces the concepts of logic programming and how it can be applied to natural languages. Definite-clause grammar is explained in section 5. The proposed model is discussed in section 6. System implementation is discussed in section 7. Sections 8 and 9 draw a number of conclusions and suggest new future research directions

2 II. Related Work

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form-generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation. Stemming programs are commonly referred to as stemming algorithms or stemmers.

A stemmer for English, for example, should identify the string ”cats” (and possibly ”catlike”, ”catty” etc.) as based on the root ”cat”, and ”stems”, ”stemmer”, ”stemming”, ”stemmed” as based on ”stem”. A stemming algorithm reduces the words ”fishing”, ”fished”, and ”fisher” to the root word, ”fish”. On the other hand, ”argue”, ”argued”, ”argues”, ”arguing”, and ”argus” reduce to the stem ”argu” (illustrating the case where the stem is not itself a word or root) but ”argument” and ”arguments” reduce to the stem ”argument”

Many areas of natural language syntax and semantics are a fruitful source of inspiration for computer languages and systems designers. The complexity of natural language and the high level of abstraction of most linguistic and semantic theories have motivated the emergence of highly abstract and transparent programming languages. One of the most striking examples is undoubtedly Prolog, initially designed for natural language parsing, via Metamorphosis Grammars (Colmerauer 1978).

For a few years, the Logic Programming paradigm has been augmented with a number of technical and formal devices designed to extend its expressive power. New logic programming languages have emerged, several of them motivated by natural language processing problems. Among them let us mention: CIL (Mukai 1985) designed to express in a direct way concepts of Situation Semantics, MOLOG (Farinas et al. 1985), an extension to Prolog designed to specify in a very simple and declarative way the semantics of modal operators and ~, -Prolog (Nadathur and Miller 1988), designed to deal with X-expressions and X-reduction.

3 Volume XVI Issue I Version I

Year 2016

4 ( )

The Logic Programming paradigm has been augmented with the concept of constrained logic programming (CLP). The basic research done within this area amounts to specifying tools for a more refined control on the type of values or terms a variable in a program can stand for. Answers to goals can be intentional: they are sets of equations (constraints) rather than mere values. Furthermore, the idea at the operational level, incorrect assignments are filtered out as soon as they are encountered when building a proof, making thus proof procedures more efficient.

The first step of any language processing system is necessarily recognizing and identifying individual words in the text. The mechanism used to generate individual words must be based on word morphology. Morphology systems can be used to decompose words into word stems and word affixes. In addition, such systems can be used to specify mood, gender, number, and person.

Many systems have been designed to address this issue. For example, Hegazi, & El-Sharkawi (1986) developed a system that detects the roots of Arabic words. This system is used to detect and correct mistakes in spelling and vowelization. Another example is a morphological analysis and generation system that is used to examine the input word for different word types and attempts to find all possible analyses **??**Saliba, & Al-Danan, 1989, Mayfield 2001).

Haddad (Haddad al et. 2005) claimed that research on computational Arabic is limited compared with English and European languages. For the last two decades, Arabic language received extensive focusing in the fields of morphological and syntactical with little attention on semantics and on deep analysis of its structures.

One of the major breakthroughs in the field of morphology was the two-level morphology. It is a general computational model for word-form recognition and generation (Koskenniemi, 1983). Lauri Karttunen and others produced a LISP implementation of the twolevel morphology and named it KIMMO (Karttunen et al. 1992). The KIMMO model consists of two components: 1. The rule component; 2. The lexical component (lexicon) (Antworth, 1990). This system had a serious deficiency: It could not determine the part of speech of a word or its inflectional categories (Antworth 1992;Xu 2002).

Al-Shalabi and Evens designed a computer system for Arabic morphology that employs a new and fast algorithm to find roots and patterns for verb forms and for nouns and adjectives derived from verbs (Al-Shalabi & Evens, 1998, Young-Suk 2003). For languages other than Arabic, a morphological syntax interface was proposed that separates syntactic function from morphological information in sequence projection architecture for the

French language. This system was designed by Frank and Zaenen (2000). A number of morphological systems, based on finite-state analysis, have also been developed by Beesley (1996Beesley (, 1998aBeesley (, 1998b)).

5 III. Lexical Analysis

Lexical analysis is the process of converting an input stream of characters into a set of words or tokens. Tokens are groups of characters with collective significance. Lexical analysis is the first stage of information gathering and natural language understanding.

The heart of a lexical analyzer generator is its algorithmic approach for producing a finite state machine. The algorithm presented in this paper is based on building finite automata with the minimum number of deterministic finite states using types of regular expressions adapted for lists of strings. During machine generation, the algorithm labels each state with the set of strings the machine would accept if that state were Year 2016

6 Global Journal of Computer Science and Technology

Volume XVI Issue I Version I ( ) the initial state. It is easy to examine these state labels to determine:

? The transition out of each state, ? The target state for each transition, and ? The states that are final.

To familiarize the reader with state labels, we provide examples that demonstrate the viability of the finite automaton mechanisms.

Case Study 1: Suppose a state is labeled with the set of strings {a, an, any, and, in, into, to, too, many, more, most}. This state must have transitions on {a, i, t, and m}. The transition on a must go to states labeled with the set { n, nd, ?}, the transition on i goes to states labeled { n, nto } ,..., and the transition on t goes to states labeled {o,oo} as in Figure 2 (Frakes & Baeze-Yates 1992).

Figure 2 shows a typical finite state machine that can be used by a lexical analyzer algorithm where an initial states q 0 with an input {a, an, and, any} can produce a new state q i , which could be a final state. In this case, the input left is ? {?, n, nd, ny}. Once the traversal reaches q 11 then nothing of the specified input is left, i.e. {?}. The input is exhausted in a similar way if one travels along the edge q 0 ? q 6 ? q 10 ? q 11

In other words, if we start reading the letter t at q 0 then control will be transferred to state q 6 with unexhausted input {o, oo}. If an o is read then this takes us to q 10 which is an accepting state with input {?, o}. If no more input characters are read then input is terminated. If more input characters are available then we traverse the arc labeled {o} to state q 11 which is a final state, i.e., input is terminated. The initial state q 0 , of the finite machine, is labeled with the following set of strings?????ï º?”{? ? ????? ? ???????ï º?”? ? ??????? ? ?????? ? ?????ï º?”? ?.}.??????

This set corresponds to {mosque, university, collect, total, sum, group of, summed up}. We want to generate all words of the set from the morpheme ????? ” ”. Figure 3 displays this example using a finite automaton representation. The morpheme ” ??????? ” , for example, can be constructed by the following arcs from q 0 ? q 1 ? q 2 ? q 4 ? q 5 ? q 4 ? q 6 . This generation is applicable for the family of morphemes listed below:

7 Volume XVI Issue I Version I

?????ï º?”? ? ????? ? ???????ï º?”? ? ??????? ? ?????? ? ?????ï º?”? ? ??????

If the building blocks of natural language texts are words, then words are important units of information, and language-based applications should include some mechanism for registering their structural properties. Finite state techniques have long been used to provide such a mechanism because of their computational effectiveness, and because of their inevitability. They can both be used to generate morphologically complex forms from underlying representations, and parse morphologically complex forms into underlying representations (Indurkhya & Damera 2010), Case Study 3:

This case study demonstrates how a variety of strings can be generated using a simple finite automaton. For example, the string ?????”? ”in Figure 4, can be generated by the following arcs (reading from right to left) 0 ? 1 ? 5 ? 12 ? 16.

8 IV. Logic Programming and Natural Languages

The rule-based approach has successfully been used in developing many natural language processing systems. Systems that use rule-based transformations are based on a core of solid linguistic knowledge. The linguistic knowledge acquired for one natural language processing system may be reused to build knowledge required for a similar task in another system.

The motivations of the rule-based approach over the corpus-based approach a: 1. Less-resourced languages, for which large corpora, possibly parallel or bilingual, with representative structures and entities are neither available nor easily affordable, and 2. For morphologically rich languages, which even with the availability of corpora suffer from data sparseness.

These have motivated many researchers to fully or partially follow the rule based approach in developing their Arabic natural processing tools and systems. In this paper we address our successful efforts that involved rule-based approach for different Arabic natural language processing tasks (Shaalan, K., 2010).

Natural language processing may require a huge amount of storage. This storage may reduce the efficiency of a software system. Using Prolog with object-oriented programming convention can address problems of complexity.

Prolog is based on an efficient and simple proof routines (Warren, & Pereira, 1980). It has dynamic memory allocation of automatic garbage collection (Roth, 1992). This facility, in addition to relieve the programmers from the notions of memory usage, makes it possible for class hierarchies, inheritance, and message passing to be generated automatically at run time. That means, Prolog has capabilities for developing oriented programs that can be taught about classes or new relationships between existing classes.

The Prolog approach yields prototyping systems that can provide convenient methods for testing the viability of rules effectively. Although Prolog implementation may not produce fast enough systems for actual use, it provides developers with sufficient details and opportunities for efficiently designing, implementing, and testing systems (Veres and Molnar, 2010).

Using such an approach in developing software systems that analyzes natural languages can aid in producing software code in small sizes compared with conventional high-level languages. In addition, the software can be organized so that it can easily be developed, understood, and maintained. Prolog is suitable for designing definite-clause grammar by which the grammar rules of the language can be translated and then the underlying language becomes executable code in Prolog.

It is convenient to restrict attention to predicate logic programs written in clausal form. Such programs have an especially simple syntax that has the expressive power of the full predicate logic. A sentence is a finite set of clauses. A clause is a disjunction L 1 V ?V L n of literals L i which are atomic formulas P(t l , . . . , t m) or the negations of atomic formulas P(t l ..... t m), where P is a predicate symbol and t i are terms. Atomic formulas are positive literals. Negations of atomic formulas are negative literals. A term is either a variable or an expression f(t l ..... t m) where f is a function symbol and t i are terms. Constants are 0-ary function symbols. A set of clauses {C 1 .... , C n } is interpreted as the conjunction, C 1 and.., and C n . A clause C containing just the variables x 1 ,..., x m is regarded as universally quantified for all x 1 , . .., x m For every sentence S 1 predicate logic there exists a sentence S 2 in clausal form which is satisfiable if and only if S 1 is exist. For this reason, all questions concerning the validity or satisfiability of sentences m predicate logic can be addressed to sentences in clausal form. Methods for transforming sentences into clausal form are described in (Nilsson, 1971). We have defined that part of the syntax of predicate logic which is concerned with the specification of well-formed formulas.

We know that we can often make "generateand-test" more efficient by pushing the test closer to the generation. How can we do this in the current situation? We do this by letting predicates like noun perform both the recognition and the splitting. We do this by letting them accept the front of a list, and return the rest of the list (Kautz , 2004)

9 V. Definite-Clause Grammars

The fundamental principle of normal language theory is that a language can be described in terms of how its sentences are constructed (Colmeraurer, 1975). That is:

1. A sentence is a string (a sequence) of symbols defined by rules for strings 2. A language is a set of sentences defined by rules for sets.

According to the above definition, we can define a grammar as: a collection of rules for specifying what sequences of symbols are acceptable as sentences (statements) of that language.

Computer scientists have adapted the ideas of formal language theory to the study of natural languages, in the form of context-free grammars (CFGs). In CFGs the basic symbols or words of the language that they describe are identified by terminal and non-terminal symbols. The terminal symbols are basic constructs of the language. The non-terminal symbols can be factorized into terminal and/or nonterminal symbols. Colmeraurer and Kowalski describe a method to translate special purpose formalism CFGs into a general one in the form of first-order predicate logic **??**Colmerauer, 1975;Kowalski, 1974; **??**aren & Pereira, 1980). The method is known as a Definite Clause Grammar (DCG). According to DCGs, rules of a grammar describe which strings of symbols are valid statements of the language.

Parsing a rule of DCGs, using Prolog, is accomplished by transforming it into a theory and trying to prove its validity by applying logical reasoning. The proof either fails or succeeds. Pereira and Warren explain the efficiency of DCGs as follows:

"If a CFG is expressed in definite clauses according to the Colmeraurer-Kowalski method, and executed as a Prolog program, the program behaves as a efficient top-down parser for the language that CFG describes. This fact becomes particularly significant when coupled with another discovery that the technique for translating CFGs into definite clauses has a simple generalization, resulting in a formalism far more powerful than CFGs, but equally amenable to execution by Prolog".

According to the Colmeraurer-Kowalski claim, the definite-clause grammar mechanism is suitable for building a logic-based framework for computational linguistics.

10 VI. A Logic-Based Framework for

Inflected Language Words

There are several types of stemming algorithms with different performance and accuracy. The various algorithms are characterized by how certain stemming obstacles are overcome.

A simple stemmer algorithm looks up the inflected form in a lookup table. The advantages, of this algorithm, are simple, fast, and easily handle exceptions. The disadvantages are that all inflected forms must be explicitly listed in the table: new or unfamiliar words are not handled, even if they are perfectly regular (e.g. iPads ~iPad), and the table may be large. For languages with simple morphology, like English, table sizes are modest, but highly inflected languages like Arabic may have hundreds of potential inflected forms for each root. A lookup approach may use preliminary part-of-speech tagging to avoid over stemming **??**Alhanini & Abo Aziz 2011).

The lookup table used by a stemmer is generally produced semi-automatically. For example, if the word is "run", then the inverted algorithm might automatically generate the forms "running", "runs", "runned", and "runly". The last two forms are valid constructions, but they are unlikely.

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm, given an input word form, in order to find its root form. Some examples of the rules, from English texts for ease of readability only, include:

? If the word ends in 'ed', remove the 'ed', ? If the word ends in 'ing', remove the 'ing', ? If the word ends in 'ly', remove the 'ly'. Suffix stripping algorithms enjoy the benefit of being much simpler to maintain than brute force algorithms, assuming the maintainer is sufficiently knowledgeable in the challenges of linguistics and morphology and be able to encoding suffix stripping rules. Suffix stripping algorithms are sometimes regarded as crude given the poor performance when dealing with exceptional relations (like 'ran' and 'run').

The solutions produced by suffix stripping algorithms are limited to those lexical categories which have well known suffixes with few exceptions. This, however, is a problem, as not all parts of speech have such a well formulated set of rules.

Suffix stripping algorithms may differ in results for a variety of reasons. One such reason is whether the algorithm constrains whether the output word must be a real word in the given language. Some approaches do not require the word to actually exist in the language lexicon (the set of all words in the language). Alternatively, some suffix stripping approaches maintain a database (a large list) of all known morphological word roots that exist as real words. These approaches check the list for the existence of the term prior to making a decision. Typically, if the term does not exist, alternate action is taken. This alternate action may involve several other criteria. The non-existence of an output term may serve to cause the algorithm to try alternate suffix stripping rules.

It can be the case that two or more suffix stripping rules apply to the same input term, which creates an ambiguity as to which rule to apply. The algorithm may assign (by human hand or stochastically) a priority to one rule or another. Or the algorithm may reject one rule application because it results in a nonexistent term whereas the other overlapping rule does not. For example, given the English term friendlies, the algorithm may identify the ies suffix and apply the appropriate rule and achieve the result of friendl. friendl is likely not found in the lexicon, and therefore the rule is rejected (Dolamic, et al. 2007).

One improvement upon basic suffix stripping is the use of suffix substitution. Similar to a stripping rule, a substitution rule replaces a suffix with an alternate suffix. For example, there could exist a rule that replaces ies with y. How this affects the algorithm varies on the algorithm's design. To illustrate, the algorithm may identify that both the ies suffix stripping rule as well as the suffix substitution rule apply. Since the stripping rule results in a non-existent term in the lexicon, but the substitution rule does not, the substitution rule is applied instead. In this example, Friendlies becomes friendly instead of friendl.

An intelligent framework based the above algorithms and on logic programming (logic-based) which enable deriving stems from inflected words (inflected words in Arabic language may form a complete meaningful sentence such as " ???????????? "

. The word ?"????????????"? is a meaningful sentence which can be factorized into ??? + ??? ??? ??? + ??? + ????? + ???? + ??? + " " which can be written as: prefix* + stem + postfix* where prefix* and postfix* is a regular expression repeated zero or more times. The framework also provides semantics of words and resolves ambiguity. It also determines the position for each addition (prefix, infix, or postfix) or bound morpheme and whether it is a subject, object or anything else. Position identification or position expression ?)??????(? is a vital necessity for enhancing understandability mechanisms. Our system is a bidirectional approach. It can deduce morphemes from inflected words or it can build inflected words from stems. The proposed software system is based on Definite Clause Grammar where rules are built according to patterns. Table **??** shows a sample of inflected morphemes. For example, the inflected morpheme ?????ï¿?"? can be identified as a pattern (templates) of the form " ?ï»?"???ï¿?"? " which has two additions: infix and postfix. The infix is " ??? " the postfix is " ?ï¿?"? " and the stem is ????? according to a pattern of the form " " ?ï»?"??? .

The finite automaton in figure 5 shows how rules can be used to derive morphemes when arcs are traversed in either direction. Where suffixes* is a regular expression that can be repeated zero or more times. It could be a pronoun and/or any other additives.

Previous researchers either store all inflected words in a lexicon, which is impractical and unrealistic, or store meaningful stems. The proposed approach is based on special patterns (templates). It associates meaning with the basic roots in order to deduce morpheme meanings. As a result, when a text file is read, stems, bound morphemes, meaning, and positions are deduced. The architecture of the framework is outlined in figure (6).

The architecture of the framework is made up of a number of components. The dialog accepts a text and then passes it to the lexical analyzer in order to decompose the text into a list of tokens.

Morphological facts are usually represented as a set of features expressed as attribute value pairs, for example, number is equivalent to singular, tense is equivalent to past participle and so on. Association of morphological features has the notion of agreement, where the form of one word depends on the features of another, or elements of a certain constituent may share certain features.

This structure can be used either to generate the appropriate inflected forms from the base forms and their feature specifications, or to give an analysis of the character strings in the reverse direction. Although our system deals only with the morphology for verbs and nouns derived from verbs or in other words verb sentences, it can easily be extended to incorporate other morphemes that are not derived from verbs, which we will address in our future work.

11 VII. System Implementation

The system has been implemented using PDP Prolog running on an IBM-compatible machine. A number of experiments have been conducted and the average has been computed. Table 2 records some findings. The beauty of logic implementation is its ability to express output in a format readable by humans and by machines. It is possible to write rules that can be fired when outputs are required to be read automatically by computer programs.

The logic rules below provide bi-directional morphological analysis. Below is a small fragment of the Prolog program. For example, the first rule run is an abstracted predicate with two arguments. The first one is "list" which is an input argument. The second argument is "W_rest" which is an output argument returning a word or parts of a word that has not been recognized. List is then factorized into five positions; L1, ?,?? , L3, L4, and the rest is R. If an ??? ?"???"? is found in the second place and, the word has more than four positions, then this predicate will be processed. Otherwise control will backtrack to the next predicate until a match has been found or failure has occurred. A sample of predicates in first-order logic that deals with inflected Arabic words is outlined below. There are several predicates, each deals with different alternative, as shown below. run (List, W_rest):- The sentences accepted by finite automata are regular sentences. In other words, there exists a finite automaton FA that accepts S(r) for any regular expression r. The structure of the proposed framework accepts S(r) in O (|r|) time and space. Although minimization is not considered, an algorithm can be constructed to minimize a deterministic FA with n states in O(|?|n log n). A number of queries is listed in Appendix I.List = [L1, ?,"?"? L3, L4 | R], List1 = [L1, ?,"?"? L3, L4], collect(List1, W_rest), chk1(R, Type),!, write(Type). run(List, W_rest):- List = [L1, ?,"?"? L3, L4], List1 = [L1, ?,"?"? L3, L4], collect(List1, W_rest). run(List, W_rest):- List = [L1, ?,"?"? L3], List1 = [L1, ?,"?"? L3], collect(List1, W_rest). run(List, W_rest):- List = [L1, L2, ?"?"? | R], List1 = [L1, L2, ?"?"?], collect(List1, W_rest), chk1(R, Type),!, write(Type). run(List, W_rest):- List = [L1, ?,"?"? ?"??"? | R], List1 = [L1, ?"?"? , ?"?"?], collect(

12 VIII. Future Works

The intelligent framework is expected to facilitate converting natural language chunks of text into more formal representations such as definite-clause grammar structures that are easier for computer programs to manipulate trough the logic programming implementation. This facilitation will involve the identification of a specific semantic from multiple ones. These identifications can be derived from natural language expressions which take the form of organized notations of natural language concepts. The framework may be extended in future research so that be able to convert information from computer storage into readable human language form.

The future research should concentrate on devising methods for inducing transformation rules that map natural-language sentences into a formal query or command language. The approach assumes a formal grammar for the target representation language and learns transformation rules that exploit the non-terminal symbols in this grammar (Kate et al. 2005;Gildea et al. 2002). The learned transformation rules incrementally map a natural language sentence or its syntactic parse tree into a parse-tree for the target formal language.

The future work may include also an intelligent interface within the proposed framework to derive high quality items of information through the process of devising patterns and trends using statistical pattern learning.

13 IX. Conclusions

The logic programming approach has successfully been used in developing many natural language processing systems. Systems that use logic programming transformations are based on a core of solid linguistic knowledge. The linguistic knowledge acquired for one natural language processing system may be reused to build knowledge required for a similar task in another system. The advantage of the logic programming approach over the corpus-based approach is for less-resourced languages, for which large corpora, possibly parallel or bilingual, with representative structures and entities are neither available nor easily affordable, and for morphologically rich languages, which even with the availability of corpora suffer from data sparseness.

These have motivated many researchers to fully or partially follow the logic programming approach in developing their Arabic natural processing tools and systems. In this paper we address our successful efforts that involved rule-based approach for Arabic natural language processing tasks. The proposed system has been

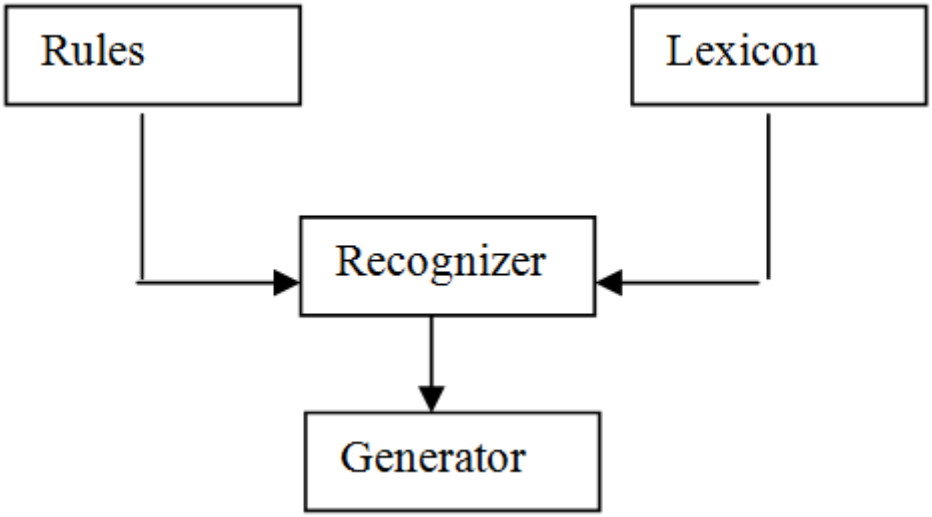
developed for deriving stems from inflected words using the logic programming language Prolog. The suggested design was based on: ? Knowledge-based mechanism embodying facts and rules, ? Inference mechanism uses the knowledge base, and a query mechanism initiated by users. The intelligent framework is used to facilitate the analyses and understanding strings from natural language. The texts are first tokenized in order to identify patterns of characters in the stream and to produce a stream of words or tokens. The tokenized text is then parsed to recognize syntactic objects according to Arabic language grammar rules.

The proposed system is a step in the direction of analyzing and understanding natural language texts. It is also potentially useful for enhancing automatic translation (Domain Specific). We conclude that there is a good case to be made from the adaption of expert systems to be used for natural languages processing. This work opens the door for more multilingual stemming research that applies morphological rules of two or more languages simultaneously instead of rules for one single language when interpreting a search query. [1] [2] [3]

1

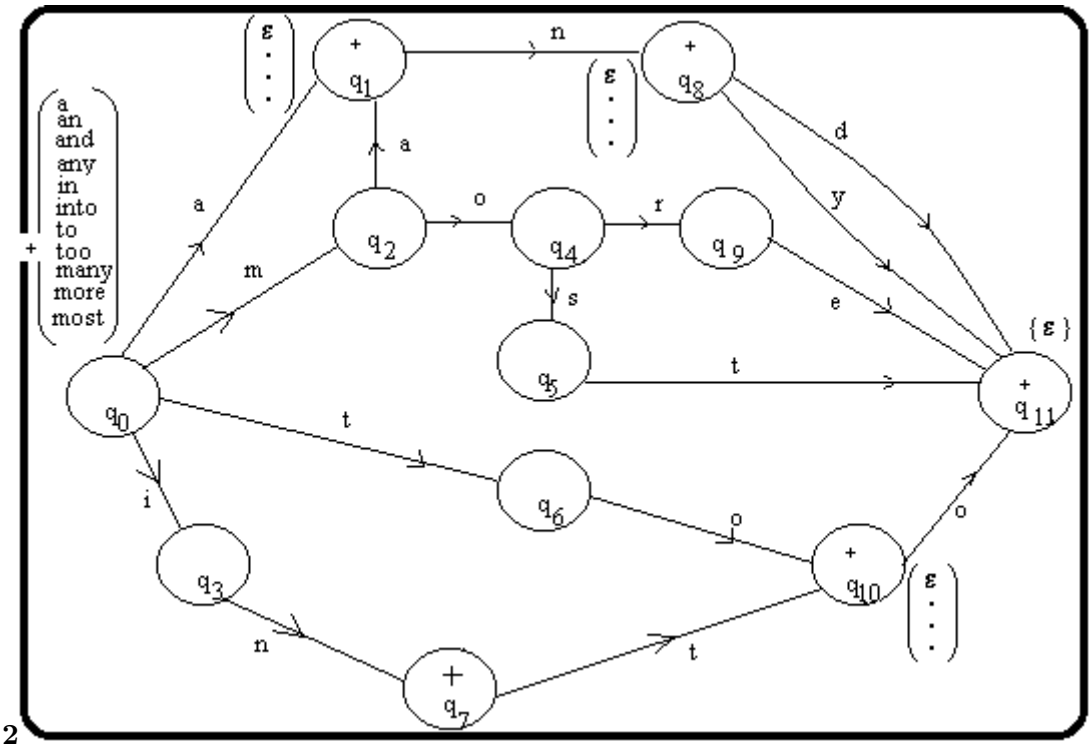
Figure 1: Figure 1

[1] © 2016 Global Journals Inc. (US)Global Journal of Computer Science and Technology
[2] © 2016 Global Journals Inc. (US) 1
[3] © 2016 Global Journals Inc. (US)



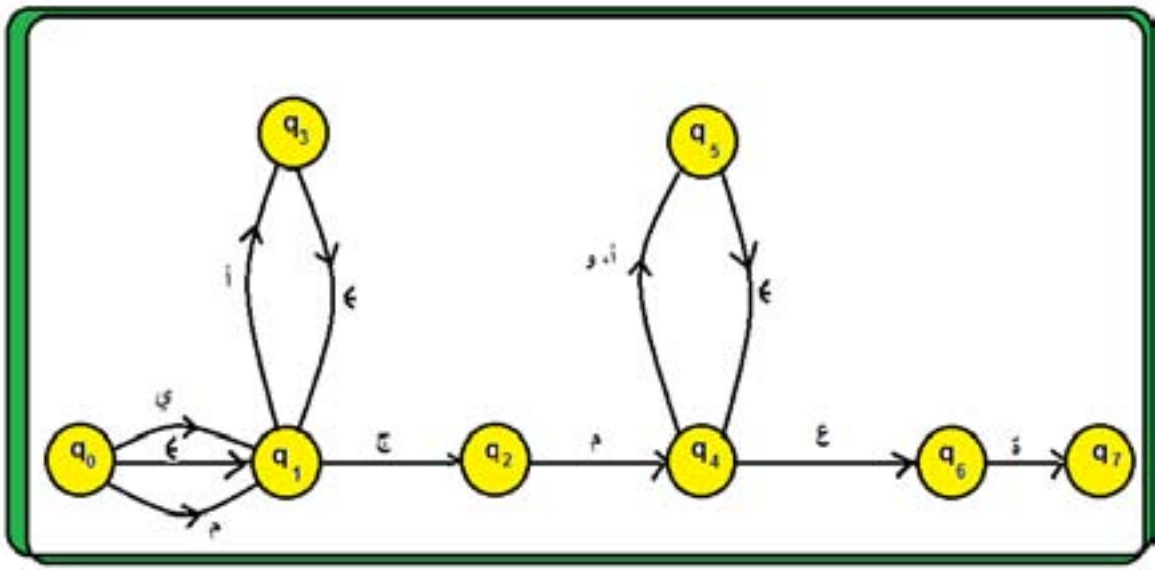
1

Figure 2: Figure 1 :



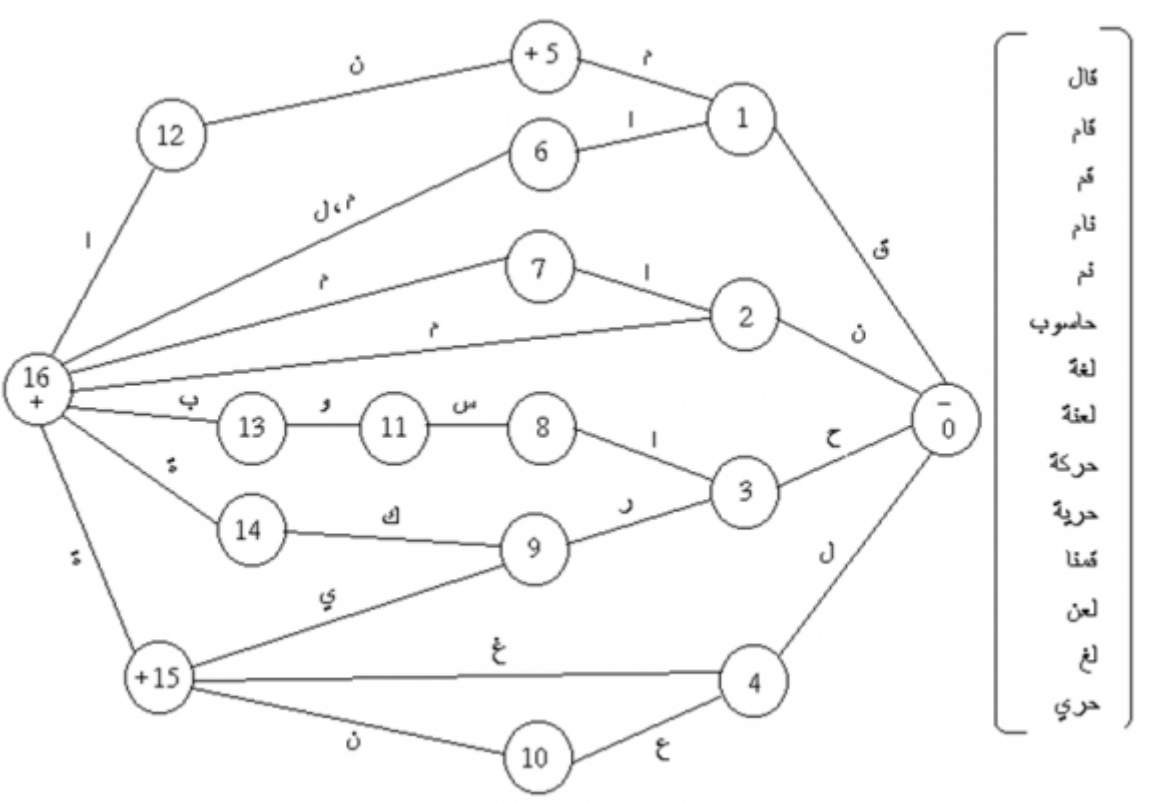
2

Figure 3: Figure 2 :



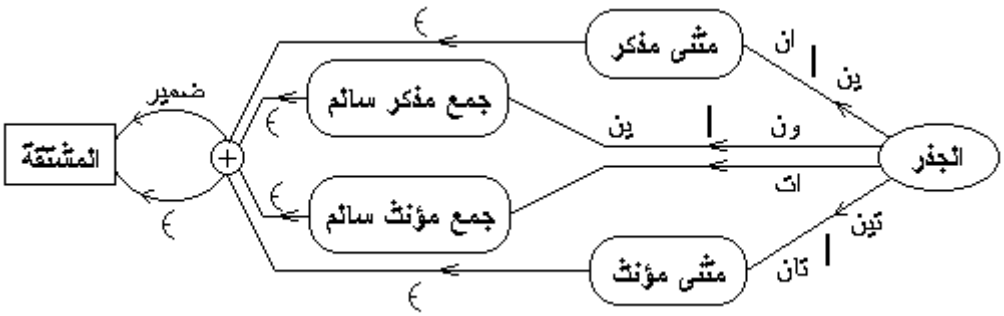
3

Figure 4: Figure 3 :



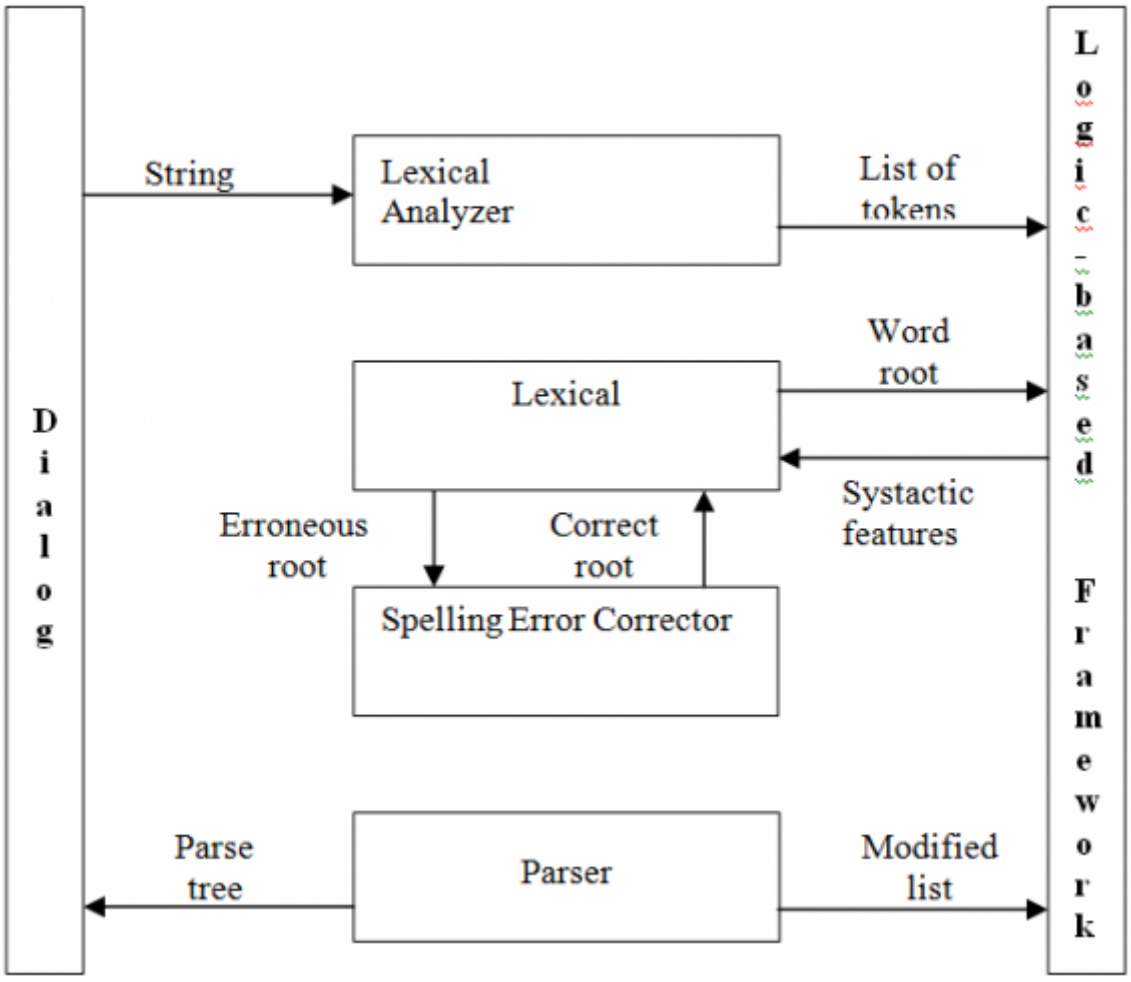
4

Figure 5: Figure 4 :



5

Figure 6: Figure 5



5

Figure 7: Figure 5 shows

l

Inflected morphemes	Postfix	Infix	Prefix		Stem
???????			?????		?????
???????ï¿?”?	?ï¿?”?		??? ?????	+	?????
??????			???		?????
??????		???			?????
????? ? ????	????				??? ? ????
?????ï¿?”?	?ï¿?”?	???			?????
????ï»?”??			???		???ï»?”??

[Note: In general, the inflected morphemes are modeled by definite-clause grammar rules as follows:inflected-morphemes ? prefix + stem | prefix + stem + postfix |stem + infix | stem + infix + postfix | prefix + stem + infix + postfix]

Figure 8: Table l :

2

1	6554 words extracted from a book	93.25%
2	5466 words extracted from newspaper	93.21%
3	2269 words extracted from Quran	90.73%

Figure 9: Table 2 :

.1 Appendix i

This appendix demonstrates the viability of the framework. Once the program has been executed a message is issued to users to enter a term (word) identity or a file name that the system will read from. When a word is entered, the framework will analyze words and break them down into their constituents.

Example 1: Enter a word? ???? ? ????????? ???? When it is translated into English, it means, "we will let them to know".

[Lee et al. ()] , Young-Suk Lee , Kishore Papineni , Salim Roukos , Ossama Emam , Hany Hassan . 2003.

[Haddad and Yaseen (2005)] 'A Compositional Approach towards Semantic Representation and Construction of Arabic'. B Haddad , M Yaseen . *the proceedings of the 5 th International Conference, LACL*, P Blache, E Stabler (ed.) (Bordeaux, France; Berlin Heidelberg) 2005. April 28-30, 2005. Springer-Verlag. p. . (Published by)

[Al-Shalabi and Evens ()] 'A Computational Morphology System for Arabic'. R Al-Shalabi , M Evens . *Computational Approaches to Semitic Languages Workshop, COLING 98*, (Montreal, Canada) 1998. p. .

[Chu-Carroll et al. (2002)] 'A multi-strategy and multi-source approach to question answering'. J Chu-Carroll , J Prager , C Welty , K Czuba , D Ferrucci . *Eleventh Text Retrieval Conference*, (Gaithersburg, MD) 2002. November. NIST Special Publications. 500 p. 251. National Institute of Standards and Technology

[Mayfield et al. ()] 'A. JHU/APL at TREC 2001: Experiments in filtering and in Arabic, video, and web retrieval'. J Mayfield , P Mcnamee , C Costello , C Piatko , Banerjee . *TREC*, (Gaithersburg) 2001. 2001. 2001. NIST.

[Hammouri ()] 'An Arabic Lexical Database to Support Natural Language Processing'. A Hammouri . *Unpublished Ph.D. dissertation* 1994.

[Nadathur and Miller ()] *An overview of 2.-Prolog*, G Nadathur , D Miller . MS-CIS-88-40. 1988. University of Pennsylvania (Technical report)

[Beesley ()] 'Arabic Finite-State morphological analysis and penetration'. K Beesley . *the 16 th International Conference on Computational Linguistics*, 1996. 1 p. . (COLLING'96)

[Beesley ()] 'Arabic morphological analysis on Internet'. K Beesley . *ICEMCO-98, Proceedings of the 6 th International Conference and Exhibition on Multilingual Computing*, 1998a. (3.1.1)

[Beesley ()] 'Arabic morphology using only finite-state operation'. K Beesley . *Proceedings of the workshop*, Michael Rosner (ed.) (the workshop) 1998b. p. . (computational approaches to Semitic languages)

[Kelley ()] *Automata and Formal Languages*, D Kelley . 1995. Englewood Cliffs, NJ: Prentice-Hall.

[Gildea and Jurafsky ()] 'Automated labeling of semantic roles'. D Gildea , D Jurafsky . *Computational Linguistics* 2002. 28 (3) p. .

[Saliba and Al-Dannan (1989)] 'Automatic Morphological Analysis of Arabic: A study of content word analysis'. B Saliba , A Al-Dannan . *Proceeding of the first Kuwait Computer Conference*, (eeding of the first Kuwait Computer ConferenceKuwait) 1989. Mar. p. .

[Chen and Sharp ()] *Content-rich biological network constructed by mining Pub Med abstracts*, H Chen , B M Sharp . 2004. p. . (BMC Bioinformatics5)

[Veres and Molnar ()] 'Documents for Intelligent Agents in English'. S M Veres , L Molnar . *Proc. AIA2010*, (AIA2010) 2010. p. 10.

[Xu et al. ()] *Empirical studies in strategies for Arabic retrieval*, J Xu , A Fraser , R Weischedel . 2002. 2002. 2002. Tampere, Finland: ACM.

[Frakes and Baeza-Yates ()] W Frakes , R Baeza-Yates . *Information Retrieval: Data Structures & Algorithms*, 1992. Prentice-Hall.

[Foss (ed.) ()] *Framing the Study of Visual Rhetoric: Toward a Transformation of Rhetorical Theory*, S Foss . Defining Visual Rhetorics. Ed. Charles A. Hill and Marguerite Helmers (ed.) 2004. Mahwah, New Jersey: Lawrence Erlbaum. p. .

[Frank and Zaenen ()] A Frank , A Zaenen . *Tense in LFG: Syntax and Morphology. To appear in Hans Kamp and Uwe Reyle: "Tense and Aspect Now*, 2000.

[Antworth ()] 'Glossing Text with the PC-Kimmo Morphological Parser'. E L Antworth . *Computers & Humanities* 1992. 26 (6) p. .

[Doms and Schroeder ()] *GoPubMed: exploring PubMed with the Gene Ontology. Nucleic Acids Research33*, A Doms , M Schroeder . 2005. p. .

[Hegazi and El-Sharkawi ()] N Hegazi , A El-Sharkawi . *Natural Arabic Language Processing, Proceeding of the 9 th National, Computer Conference and Exhibition*, (Riyadh, Saudi Arabia) 1986. p. .

[Indurkhya ()] N Indurkhya , F . *Handbook of Natural Language Processing*, Francis Taylor, Group, Ll (ed.) 2010. (2 nd edition)

[Can et al. ()] 'Information retrieval on Turkish texts'. F Can , S Kocberber , E Balcik , C Kaynak , H C Ocalan . *Journal of the American Society for Information Science and Technology* 2008. 59 p. .

[Hopcroft and Ullman ()] *Introduction to Automata Theory*, J Hopcroft , J Ullman . 1979. Reading Mass: Addison-Wesley.

[Language Model Based Arabic Word Segmentation, for Computational Linguistics Proceedings of the 41st Annual Meeting of th 'Language Model Based Arabic Word Segmentation, for Computational Linguistics'. *Proceedings of the 41st Annual Meeting of the Association*, (the 41st Annual Meeting of the Association) July 2003. p. .

[Kate et al. (2005)] 'Learning to Transform Natural to Formal Languages'. R J Kate , Y W Wong , R J Mooney . *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, (the Twentieth National Conference on Artificial Intelligence (AAAI-05)Pittsburgh, PA) 2005. July 2005. p. .

[Colmeraurer ()] *Les Grammaires de Metamorphose*, A Colmeraurer . 1975. Groupe d'Intelligence Artificielle, University' de marseille-Luminy

[Colmerauer ()] *Metamorphosis Grammars, in: Natural Language Understanding by Computer, Lecture notes in Computer Science*, A Colmerauer . 1978. 1978. Springer-Verlag. L. Bole Edt

[Del Cerro and Arthaud ()] *Molog: Programming in Modal Logic, Fifth Generation Computing journal*, Farinas Del Cerro , L Arthaud , A . 1985. 1985.

[Kautz (2004)] *Natural Language Understanding*, H Kautz . https://courses.cs.washington.edu/courses/csep573/04au/lectures/nlp-all.pdf 2004. July 2016.

[Antworth ()] *PC-KIMMO: A two-level processor for morphological analysis. Number 16 in occasional publications in academic computing*, E L Antworth . 1990. Dallas. Summer Institute of Linguistics

[Kowalski ()] 'Predicate Logic as Programming Language'. R Kowalski . *Proc. IFIP 74*, (IFIP 74Stockholm) 1974.

[Nilsson ()] *Problem Solving Methods m Artificial intelligence*, N J Nilsson . 1971. New York: McGraw-Hill.

[Bouma et al. ()] 'Question answering with joost at CLEF'. G Bouma , J Mur , G Van Noord , L Plas , J Tiedemann . *Workshop of Cross-Language Evaluation Forum (CLEF2008)*, (Aarhus, Denmark) 2008. 2008.

[Roth ()] A Roth . *Prolog a Better Bet Than C++? Program Now*, 1992. p. .

[Shaalan (2010)] 'Rule-based Approach in Arabic Natural Language Processing'. K Shaalan . *International Journal on Information and Communication Technologies* 2010. June 2010. 3 (3) .

[Saharia et al. ()] N Saharia , K M Konwar , U Sharma , J K Kalita . *An Improved Stemming Approach Using HMM for a Highly Inflectional Language, Computational Linguistics and Intelligent Text Processing*, the series Lecture Notes in Computer Science 2013. 7816 p. .

[Dolamic and Savoy ()] *Stemming Approaches for East European Languages*, Ljiljana ; Dolamic , Jacques Savoy . 2007. 2007.

[Jain and Agrawa ()] 'Text independent root word identification in Hindi language using natural language processing'. L Jain , P Agrawa . *International Journal of Advanced Intelligence Paradigms* 2015.

[Alhanini and Aziz ()] 'The Enhancement of Arabic Stemming by Using Light Stemming and Dictionary-Based Stemming'. Y Alhanini , Ab Aziz , MJ . *Journal of Software Engineering and Applications* 2011. 2011. 4 p. .

[Van Emden and Kowalski (2016)] *The Semantics of Predicate Logic as a Programming Language*, M H Van Emden , R A Kowalski . http://www.doc.ic.ac.uk/~rak/papers/kowalski-van_emden.pdf 2016. July 2016.

[Daimi and Abdel-Amir ()] 'The Syntactic Analysis of Arabic by Machine'. Al Daimi , K Abdel-Amir , MA . *Computers and Humanities* 1994. 28 p. .

[Karttunen and Kaplan ()] 'Two-level Morphology with composition'. L Karttunen , R Kaplan , ZaenenA . *Proceedings of the 14 th International Conference on Computational Linguistics COLLING-92*, (the 14 th International Conference on Computational Linguistics COLLING-92Nantes, France) 1992. I p. .

[Koskenniemi ()] *Two-level Morphology: a general computational model for word-form recognition and production*, K Koskenniemi . 1983. University of Helsinki (department of General Linguistics)

[Bornat ()] *Understanding & Writing Compilers*, R Bornat . 1985. Macmillan Publishers Ltd.

[Mukai ()] 'Unification over Complex Indeterminate'. K Mukai . *Fifth Generation Computer Journal* 1985.