# USING DATA-DRIVEN RESOURCES FOR OPTIMISING RULE-BASED SYNTACTIC ANALYSIS FOR MODERN STANDARD ARABIC

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2014

By
Mohamed Nassim ELBEY
School of Computer Science

# Contents

# List of Tables

# List of Figures

# Abstract

This thesis is about optimising a rule based parser for Modern Standard Arabic (MSA). If ambiguity is a major problem in NLP systems. It is even worse in a language MSA due to the fact that written MSA omits short vowels and for other reasons that will be discussed in Chapter 1.
By analysing the original rule based parser, it turned out that many parses were unnecessary due to many edges being produced and not used in the final analysis. The first part of this thesis is to investigate whether integrating a Part Of Speech (POS) tagger will help speeding up the parsing, or not. This is a well-known technique for Romance and Germanic languages, but its effectiveness has not been widely explored for MSA. The second part of the thesis is to use statistics and machine learning techniques and investigate its effects on the parser. This thesis is not about the accuracy of the parser. It is about finding ways to improve the speed. A new approach will be discussed, which was not explored in statistical parsing before. This approach is collecting statistics while parsing, and using these to learn strategies to be used during the parsing process. The learning process involves all the moves of the parsing (moves that lead to the final analysis, i.e good moves and moves that lead away from it, i.e bad moves). The idea here is, not only we are learning from positive data, but also from negative data. The questions to be asked:

- Why is this move good so that we can encourage it.
- Why is this move bad so that we discourage it.

In the final part of the thesis, both techniques were merged together: integrating a POS tagger and using the learning approach, and finding out the effect of this on the parser.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

**Buckwalter Transliteration**

| Letter | Trans | Name | Letter | Trans | Name |
|---|---|---|---|---|---|
| ء | ’ | Hamza | ظ | Z | Dhah |
| آ | \| | Alif madda | ع | E | Ain |
| أ | O | Alif hamza above | غ | g | Ghain |
| إ | I | Alif hamza below | ف | f | Feh |
| ؤ | W | Waw hamza above | ق | q | Qaf |
| ئ | } | Yeh hamza above | ك | k | Kaf |
| ا | A | Alif | ل | l | Lam |
| ب | b | Beh | م | m | Meem |
| ت | t | Teh | ن | n | Noon |
| ة | p | Teh marbutah | ه | h | Heh |
| ث | v | Theh | و | w | Waw |
| ج | j | Jeem | ي | y | Yeh |
| ح | H | HHeh | ى | Y | Alif maqsura |
| خ | x | Kheh | ◌َ | a | Fatha |
| د | d | Del | ◌ُ | u | Dhamma |
| ذ | * | Dhel | ◌ِ | i | kasra |
| ر | r | Reh | ◌ْ | o | Sukun |
| ز | z | Zain | ◌ّ | | Shadda |
| س | s | Seen | ◌ً | F | Fathatan |
| ش | $ | Sheen | ◌ٌ | N | Dhammatan |
| ص | S | Sad | ◌ٍ | K | Kasratan |
| ض | D | Dad | | | |
| ط | T | TTah | | | |

**Abbreviations and Acronyms**

| Abbreviation | Full name |
|---|---|
| BPC | Base Phrase Chunker |
| CA | Classical Arabic |
| CATiB | Columbia Arabic Treebank |
| CFG | Context Free Grammar |
| DS | Dependency Structure |
| GPSG | Generalised Phrase Structure Grammar |
| HMM | Hidden Markov Model |
| HPSG | Head-driven Phrase Structure Grammar |
| LFG | Lexical Functional Grammar |
| MLE | Maximum Likelihood Estimation |
| MSA | Modern Standard Arabic |
| NLP | Natural Language Processing |
| NP | Noun phrase |
| OVS | object-verb-subject |
| PADT | Prague Arabic Dependency Treebank |
| POS | part-of-speech |
| PS | Phrase Structure |
| S | Sentence |
| TBL | Transformation-Based Learning |
| VOS | verb-object-subject |
| VP | verb phrase |
| VSO | verb-subject-object |

# Chapter 1

# Introduction

## 1.1 Overview

Natural Language Processing NLP is a field of Artificial Intelligence that involves anything that processes written and spoken languages. It involves a range of computational techniques that analyze and represent natural languages at several levels of linguistic analysis [JM00]. These levels are:

- Phonetics and Phonology: the interpretation and the study of linguistic sounds.
- Morphology: the study of the meaningful components and analysis of words, including prefixes, suffixes and roots.
- Lexical analysis: word level analysis including lexical meaning and part of speech analysis.
- Syntactic analysis: the study of words in a sentence in order to uncover the grammatical structure of the sentence i.e. the study of structural relationship between words.
- Semantics: determining the possible meanings of a sentence, including disambiguation of words in context.
- Discourse: interpreting structure and meaning conveyed by texts larger than a sentence or a single utterance.
- Pragmatics: the study of how language is used to reach goals (understanding the purpose and use of language in situations, particularly those aspects of language which require world knowledge).

In this thesis we are interested in the syntactic and morphological analysis of Modern Standard Arabic MSA sentences. In recent years, there has been a great need for Natural Language Processing (NLP) resources in general, and particularly in MSA. The interest in MSA has become increasingly important due to the lack of NLP tools available. This research mainly aims at working with a rule based NLP engine for MSA and apply a set of experimentations and investigate its effect on the speed of this engine. This system is called PARASITE (PrAgmatics = ReAsoning about the Speaker's InTEnsions) which is a Sicstus Prolog program for performing a range of linguistic tasks on several languages including MSA. In my research I am interested in two elements of PARASITE:

- Lexical and morphological processing: A dictionary is used to improve the time of retrieving the words. There exists a fairly large dictionary for English. However, there is only a basic one of 286 entries for MSA.

- Syntax and parsing: The framework used in PARASITE is somewhere between HPSG [SW99] and pure categorial grammar [SW99] but with a rather radical approach to extraposition. This will be explained later.

The first part of my research is to incorporate a part of speech POS tagger into the parser and investigate its effects on the speed. The second part is to use machine learning techniques to learn tactics to be used during the parsing process. The third part is to use both techniques at the same time.

## 1.2 Part of Speech tagging

Part of Speech tagging (also called word-class tagging, or grammatical tagging) is the process of assigning a part of speech tag such as noun, verb, pronoun, preposition, adverb, adjective, or other tags to each word in a sentence. It reflects the word's syntactic category based on its context for the purpose of resolving ambiguity. A single word form may have more than one part of speech assigned to it. Although, most words of English are unambiguous (i.e. they have only one Brown corpus tag), many of the common words are ambiguous. That is, it has more than one possible part of speech tag. For instance the English word "book" is functioning as a verb in the sentence "I book a ticket", or as a noun in the sentence "the book that I have read". Derose

[DeR88] reports the degree of ambiguity of English in the Brown corpus as shown in Table 1.1.

| | |
|---|---|
| Unambiguous (1 tag) | 35340 |
| Ambiguous (2-7 tags) | 4100 |
| Ambiguous (2 tags) | 3760 |
| Ambiguous (3 tags) | 264 |
| Ambiguous (4 tags) | 61 |
| Ambiguous (5 tags) | 12 |
| Ambiguous (6 tags) | 2 |
| Ambiguous (7 tags) | 1 |

Table 1.1: The degree of POS ambiguity of English in the Brown corpus [DeR88]

However, the ambiguity in Arabic is much worse due to its complex morphology, the lack of diacritics (omitting the short vowels) and also Arabic being a free word order language. A simple word like علم *ʕlm* can be interpreted as عَلَّمَ *ʕallama* (taught), عُلِّمَ *ʕullima* (was taught), عَلِمَ *ʕalima* (knew), عِلم *ʕilm* (knowledge), عَلَم *ʕalam* (flag), or the imperative form عَلِّم *ʕallim* (teach).

Given a corpus of millions of words, assigning a POS tag to each word of a text by hand is very time consuming. That is why various techniques have been used to automate the tagging process. POS tagging can be used in several applications such as Text-to-Speech, Speech synthesis, Information retrieval, parsing, Machine learning, and other NLP tasks. For instance, the word ”content” can be a noun or an adjective, and they are pronounced differently. The noun is pronounced CONtent and the adjective is pronounced conTENT. Therefore, by knowing the correct tag, we can produce more natural pronunciations in speech synthesis systems. In my thesis, I am going to investigate the effects of using POS tagging with the parser and see whether it may make the parsing run faster by eliminating unnecessary parses.

The linguistic characteristics of Arabic present some interesting challenges to NLP researchers. Like many non-European languages, Arabic is lacking in annotated resources and tools, such as tokenizers and Part of Speech taggers. For languages like English, word-level POS tagging seems sufficient since words usually correspond to the syntactically relevant POS tag classes. However, for Arabic, the syntactically relevant POS tag classes do not necessarily correspond to words. Arabic words are often formed by concatenating smaller parts, each of which has its own POS tag as shown in Figure 1.1. The main part of the word is the stem which is often derived from a root morpheme. The other parts are affixes and clitics.

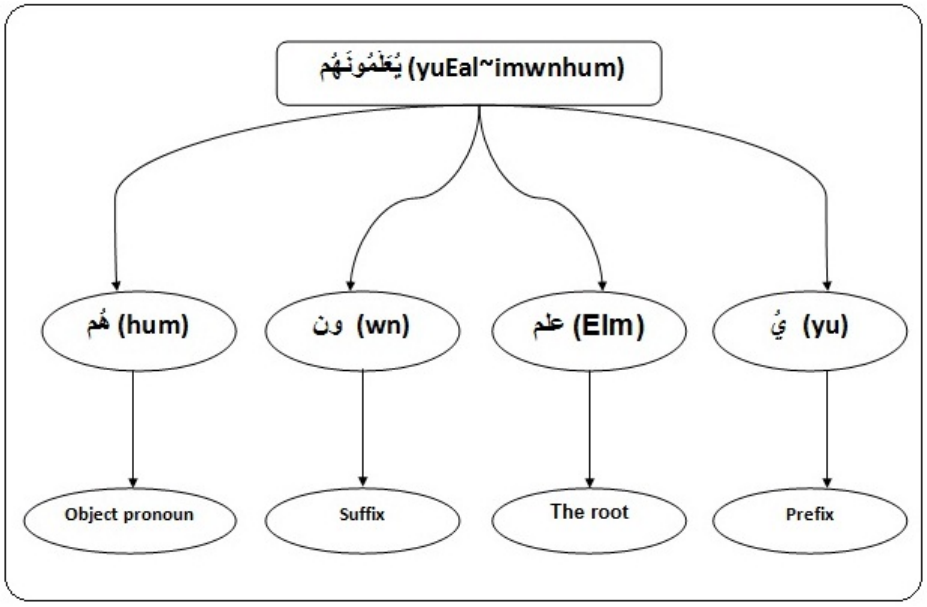



Figure 1.1: Concatenating the stem with affixes and clitics to form an Arabic word

Clitics include certain prepositions, conjunctions, determiners and pronouns. The affixes include inflectional markers for tense, gender, and number.

Incorporating a part-of-speech tagger into a system that performs linguistic analysis was not widely exploited in MSA. Therefore, the current research aims at integrating a part of speech tagger to PARASITE and investigating its effects on its speed. What is wanted from this method is to make the tagger serve as front end to the parser: the tagger assigns the tags to the incoming words, and the parser can work at the tag level. One of the early studies to address this was Charniak [CCA+96], whose work considered two approaches: single taggers and multiple taggers. Single taggers return a single best tag for each work, and multiple taggers return a list of possible tags. Charniak concluded that single taggers are better and multiple taggers do not significantly increase the accuracy relative to single taggers.

In another study, Wauschkuhn [Wau95] used a tagger as a preprocessor for a partial syntactic analysis in German corpora. He concluded that tagging helped reduce the number of parse trees but at the cost of increasing the rate of unsuccessful parses.

Another study was conducted by Voutilainen [Vou98] based on using two different rule-based morphological disambiguators as preprocessors of a finite-state parser of English. It was concluded that tagging helps reducing ambiguity but at the same time increases the number of sentences that have no parse.

In 2001 Prins and van Noord [PN01] investigated the effects of adding an unsupervised tagger trained on the output of the parser. They concluded that not only did it reduce the number of lexical categories, but also mildly increased the parsing accuracy.

Kaplan [KK03] described three types of shallow mark-up: part of speech tagging, named entities, and labelled bracketing. He concluded that named-entity mark-up improves both speed and accuracy and labeled bracketing can also be beneficial, whereas part-of-speech tags are not particularly useful. With regards to Modern Standard Arabic, it was claimed recently by Alabbas and Ramsay [AM12] that for tasks where precision is important it is worth merging a POS tagger with the parser, but you do lose almost the same level of recall as you gain in precision.

## 1.3 Statistical Parsing

Machine learning and parsing is explored in the second part. Early work was done by Brill [Bri95], Collins [Col97] and Charniak [Cha97].

Building traditional natural language parsers requires lots of human intervention, where sentences have to be manually examined, and grammar rules which cover the language are written. By observing the performance of the grammar, and analysing the errors made by this type of parsers, the rules are carefully refined. This process is then repeated over a long period. The grammar refinement process is extremely time consuming and a difficult task to do, and it did not succeed to accurately parse a large corpus of unrestricted text [JLM$^+$]. As an alternative to writing grammars, one can use treebanks to extract the grammar. One of these studies was conducted by Charniak (97) who presented a statistical parser that induced its grammar and probabilities from a hand parsed corpus, and showed that its performance is superior to the previous parsers. However, the drawback of this method, is that creating the training corpus requires lots of efforts and manual intervention.

It is necessary to have a large annotated corpus to build a statistical parser. Acquisition of such a corpus is costly and time-consuming [TLR02]. Therefore, we thought of using machine learning while parsing. The idea here is to carry out a series of parses and collect useful information that can be used in the next parses, and investigate its effects on the parsing speed.

Lots of work have been done in statistical parsing, and most of it focused on English. Several methods were used for statistical parsing such as Maximum Entropy model for phrase structure representations [Cha] and Support Vector Machine models

(SVMs) for dependency grammar [HNN+07].

Although many languages have developed sophisticated NLP systems, MSA is a particularly problematic language. Taking NLP techniques that work for English and applying them unchanged to Arabic is not going to work. The problem is that there are number of sources of ambiguity in Arabic that are not present in Romance and Germanic languages. We say some input is ambiguous if it can be interpreted in more than one way. This problem is particularly serious for MSA because of its combination of extremely complex morphology with the lack of diacritics (omission of vowels), and the comparatively free word order. All this has made Arabic a very ambiguous language, at least for computational systems. Arabic speakers, on the other hand, are able to read any MSA text without any difficulties, because of their ability to mix various levels of linguistic processing, which help them choose the right interpretations.

## 1.4 Objectives and methodology

The main objective of my project is to apply a variety of heuristics and investigate its effectiveness on the speed of a rule based engine for MSA, rather than improving its accuracy.
This engine currently explores all possibilities, which makes the system runs slowly if given a long text. Our experimentations consist of three parts:

- Integrating a part of speech tagger into the parser and investigating the effects of this on the behaviour of the parser. This is a well-known technique for Romance and Germanic languages, but its effectiveness has not been widely explored for MSA. These effects are mainly on the search space of using the tagger in various ways:
  - Using its suggestions directly (This may lead to a complete failure in some cases where the suggested tag is wrong).
  - Because tagging MSA is not likely to be 99% accurate we are using the tagger to guide the parser by introducing penalties (scores). A good score is given if the tagger and parser agrees on the tag. However if they have different tags, then a bad score is given.
- Investigating the use of machine learning techniques to learn tactics to be used during the parsing process. The aim of this part is to carry out a series of parses

and collect information regarding the moves. The moves of the parsing are divided into three types: moves that lie on the actual solution (Good moves), moves that lead directly away from it (Bad moves), and moves that follow on from bad moves (Neither good nor bad moves). This process is as follows:

- Parsing a body of text and collecting statistical facts about it (good and bad edges).
- choosing a plausible set of features.
- using Weka to learn classification rules (Weka "Waikato Environment for Knowledge Analysis" is a popular suite of machine learning software written in Java, developed at the University of Waikato. WEKA is free software available online).
- This initial set of analyses can be exploited to provide more statistical evidence, which can be used to obtain analyses of more complex sentences, which can in turn be used to continue this process again.

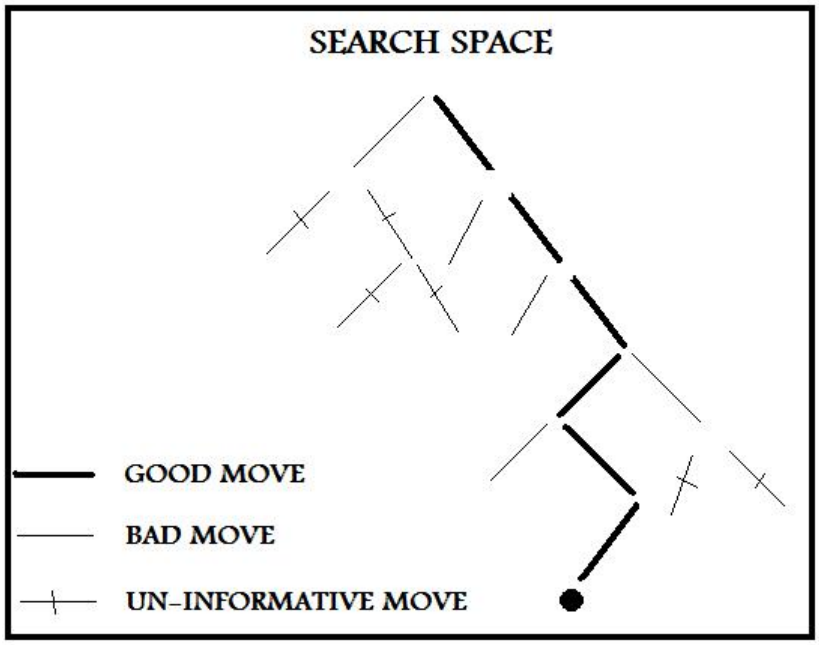



Figure 1.2: Good and bad edges

- Investigating the effects of using both the POS tagger and Machine learning at the same time.

- Investigating the extent of these techniques on the accuracy of the parser. For this, we assume that the output we get from the original parser before applying these techniques as our gold standard results.

# Chapter 2

# Background: MSA challenges

## 2.1 Overview

NLP involves several levels of linguistic analysis. It is not surprising that all these levels share one common fact which is resolving ambiguity. In this chapter, it is worth introducing the levels of ambiguity in NLP in general, and specifically for MSA. First, the problem of ambiguity in English is introduced, including lexical and syntactic ambiguity. Then, we describe the linguistic features of MSA, especially those that make the language very ambiguous.

We say some input is ambiguous if it can be interpreted in more than one way. In NLP ambiguity may occur at different levels (lexical, syntactic, and semantic).

## 2.2 Lexical ambiguity

Lexical ambiguity is a common problem in natural language processing. We talk about lexical ambiguity when a single word has two or more interpretations or can be used in different ways.

- **Homonymy vs Polysemy:** Lexical ambiguity is usually divided into polysemy and homonymy. Polysemy refers to one word having several related meanings for example the word ”line” means ”a thread”, ”a row”, etc, whereas homonymy describes the fact that two words have the same lexical form, but different etymologies and unrelated meanings for instance: the word ”bank” has several distinct definitions including ”financial institution” and ”edge of a river”. Another example is the adjective ”light” which means ”not heavy” and ”not dark”.

**Example 1:** Homonyms

1. *I hope you are not lying to me.*
2. *My books are lying on the table.*
3. *The kids are going to watch TV tonight.*
4. *What time is it? I need to set my watch.*

The word *lying* in sentence 1 and the word *lying* in sentence 2 are two words that happen to share the same sound and spelling. There is no relation between them. The word *lying* in 1 means telling a lie whereas in 2 it means being in a horizontal position. The word *watch* in 3 means to look at, whereas in 4 it means a small clock worn on the wrist.

**Example 2:** Polysemes

1. *The newspaper fired its editor.*
2. *John spilled coffee on the newspaper.*
3. *The newspaper has decided to change its format.*
4. *John used to work for the newspaper that you are reading.*

The word *newspaper* in sentence 1 means *the company that publishes the newspaper*. The same word has different related meanings: in 2, it means *the physical newspaper*, in 3, it refers to *the newspaper as an edited work* and in 4, it is about *the newspaper as an institution, a tangible object, and a piece of information*.

- **Homographs vs Homophones:** Homographs are words that have the same spelling, but different pronunciations and meanings, whereas homophones are Words that have the same pronunciation, but different spelling and different meanings.

  **Example 3:** Homographs

  1. *The wind is blowing hard.*
  2. *I have to wind my clock.*
  3. *I do not know if I will live or die.*
  4. *Last night I saw the band play live in concert.*

The word *wind* is a homograph because one written form has two different meanings: each is pronounced differently. The word *wind* in 1 means a moving air, and it rhymes with *pinned*. On the other hand the same written form *wind* in 2 means making the clock working by turning the key or handle, and it rhymes with *find*. Similarly, the word *live* is a homograph, since it is one written form representing two different meanings and pronunciations. *live* in sentence 3 means to have life and it rhymes with *give*. However in sentence 4 it means in real time performance and it rhymes with *hive*.

## 2.3 Syntactic Ambiguity

Syntactic ambiguity happens when a phrase can be parsed in more than one way. Syntactic ambiguity does not arise from the range of meanings of single words, but from the relationship between the words and clauses of a sentence.
If lexical ambiguity involves different interpretations of a word, syntactic ambiguity, however, involves different interpretations of the meaning of the whole sentence. In other words, in lexical ambiguity, the individual words are interpreted differently, but the structure remains the same, whereas in syntactic ambiguity the same sequence of words is interpreted as having different syntactic structures. One productive source of syntactic ambiguity in English is the attachment of modifiers, especially prepositional phrases.

- **PP attachment ambiguity:** Given a sequence ”VP NP PP”, should the ”PP” be attached to the main verb, or to the object noun phrase? For example, in the widely used illustrative sentence: ”I saw the man with the telescope”. This sentence is ambiguous in a way that attaching the PP to the object noun would mean that the man is in possession of the telescope, while attaching to the verb would be interpreted as seeing using the telescope. Figure 2.1 shows the two parse trees of this sentence.

  - Using a telescope I saw the man.
  - I saw the man that has the telescope.

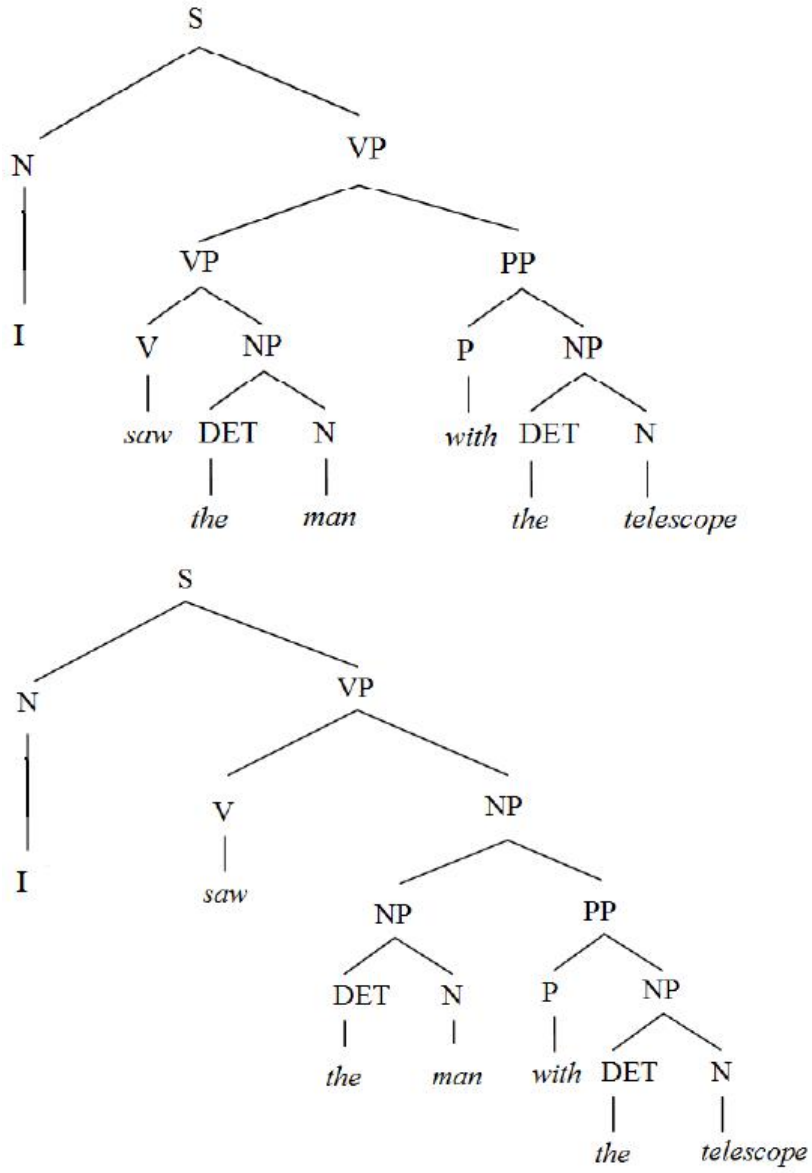



Figure 2.1: Two parse trees of the sentence *"I saw the man with the telescope"* [JM00]

## 2.4 Linguistic aspects of Arabic

### 2.4.1 Overview

Arabic is considered as one of the Semitic languages. It is spoken throughout the Arabic world and widely studied and known throughout the Islamic world. Arabic differs from English or other European languages syntactically, morphologically and semantically. A huge aspect of Arabic is that it has extremely complex morphology, made worse by the lack of written short vowels, and also its comparatively free word order.

Arabic is usually classified into three main forms, namely, Classical Arabic (CA), Modern Standard Arabic (MSA) and Colloquial Arabic. Classical Arabic is the language used in the old literature and religious books. It is the language of the Quran and Sunna (Prophetic traditions). MSA is hugely used by present-day media, in publications, and academic institutions across the Arabic world. As people in the Arabic world speak different dialects, MSA is used as the only means of communication.

The colloquial Arabic refers to the regional dialects spoken by people in the Middle East, North Africa, and other Arabic countries. MSA shares some grammatical and Syntactical features with CA, whereas Colloquial Arabic is radically different. The main differences between Classical and MSA are that MSA has a larger vocabulary, and does not use some of the more complicated forms of grammar found in Classical Arabic [Kho01]. In addition, CA is diacriticised, where we write diacritic marks above the consonants, whereas MSA is written without diacritics. This is like writing English without vowels, and it is up to the reader to give the right interpretation of the word.

Unlike in English, there are several types of vowels in Arabic:

- Long vowels: 'A', 'w'and 'y'.
- Short vowels: 'a', 'u', 'i' and they are called diacritics.
- Other symbols:
  - 'F' which is Double 'a' and its sound is 'an'
  - 'N' which is Double 'u' and its sound is 'un'
  - 'K' which is double 'i' and its sound is 'in'

– 'o' is called Sukun in Arabic where the consonant is not followed by a vowel

There exists various approaches for romanizing the Arabic text which is often termed as *transliteration*. they vary in the way they address written and spoken Arabic in the Latin alphabet. They also use different symbols for Arabic phonemes that do not exist in English or other European languages. In this thesis, Buckwalter [Buc04] Transliteration standard is used. This was developed at Xerox by *Tim Buckwalter* in the 1990s. The reason why we chose Buckwalter is that unlike other transliterations where some characters are mapped with two Latin characters, Buckwalter strictly represents every Arabic character by one Latin character. The letter ش *š* for instance, is represented as "*sh*" in Qalam transliteration, and as $ in Buckwalter transliteration.

## 2.4.2 Sources of ambiguity in Modern Standard Arabic

There are many factors that contribute to the problem of ambiguity in MSA. In this section, we investigate the different sources of ambiguity that lead to the language being very ambiguous. These factors can be described as follows:

### 2.4.2.1 Lack of diacritics or omission of short vowels

MSA consists of 28 letters which are explicitly present in the written form. However, the diacritics (short vowels) are optional. Therefore the NLP challenge is to first recover the diacritics to reduce the ambiguity.

### 2.4.2.2 Lexical Ambiguity

Due to the complex morphology and the optional diacritization of MSA words which may lead to a larger number of homographs, there is more ambiguity than may be found in English. Lexical ambiguity in MSA can be divided into the following types:

- **Homonymic words that have two meanings:** for example the word قَدَم *qadam* (meaning *foot*) or قَدَم *qadam* (meaning *unit of length*). Another example is لِسَان *lisān* (meaning *tongue* as an organ of body) and لِسَان *lisān* (meaning *language*).
- **Homonymic words that have more than two meanings:** for example the word فُصُول *fuṣuwl* has several meanings: (chapters of a book, seasons of the year,

academic semesters in school, acts of play, and more). To illustrate the extent of ambiguity in MSA, the word عَين *ʿayn* has the following meanings:

- عَين الْاِنسَان *ʿayn ālā'nsān* meaning *the eyes*.
- فيهمَا عَينَان تجرِيَان *fyhmā ʿaynān tğriyān* meaning *fountain*.
- أَلجَاسوس *alğāsws* meaning *the spy*.
- أهل البلد *ʾhl ālbld* meaning *the local people*.
- رَئِيس الجيش *raʾys ālğyš* meaning *the army's leader*.
- كبير القوم و شريفهم *kbyr ālqwm w šryfhm* meaning *the community leader*.
- أَصَابته العين *ʾṣābth ālʿyn* meaning *He caught the bad eye*.
- أغمض عينه عنه *ʾġmḍ ʿynh ʿnh* meaning *He ignored him*.
- سقط من عيني *sqṭ mn ʿyny* meaning *he has lost my respect*.
- قُرَّت عينه *qurrat ʿynh* meaning *he became happy*.

- **Homonymic words that have a relation in meaning:** This is like polysemy in English. For example the word ذِرَاع *ḏirāʿ* might mean *arm* which is an organ of body, or *a unit of measurement*.

- **Homonymic words that have no relation in meaning:** This is like homonyms in English. They have the same written form and are pronounced the same but they have different meanings. For example the word قَرن *qarn* might mean *a century*,or *a horn* for instance of a rhinoceros. Other examples like: the word سن *sn* which means *age*, and also it means *tooth*. Also the word دقيق *dqyq* means *flour*, *precise*, or *thin*.

- **Homographs:** These are words that have the same written form but have different pronunciation. The lack of diacritics makes MSA very ambiguous, where the same written form has several of interpretations for example: كتب *ktb* when filling in the diacritics we will have كَتَبَ *kataba*, كَتَّبَ *kattaba*, كُتِبَ *kutiba*, and كُتُب *kutub*.

- **Homophones:** these are words that have the same sound but are written differently. for example يَحيَا *yaḥyā* meaning *to live*, and يَحيَ *yaḥyā'* for the proper noun *Yahia*.

#### 2.4.2.3 Morphological ambiguity

This refers to the fact that a word can be ambiguous with regard to its internal structure. For example, in English the verb form ”look” can either refer to the infinitive, first or second singular, or plural. However, as soon as there are more information such as preceding words, the ambiguity can be resolved. For example ”to look” which refers to the infinitive form. Morphological ambiguity is a serious problem in MSA due to its complex morphology. Resolving this ambiguity is a major challenge for NLP Arabic researchers. Below are the factors that contribute to this problem:

- **Extremely complex derivational and inflectional morphology:** In order to have a real insight into the language, one must look for roots of words. Each inflection of the language is represented by means of infixes, prefixes and suffixes. Most words are built up from roots each of which consists of three consonants. Words are produced by following certain patterns and adding infixes, prefixes and suffixes. Table 2.1 below shows the conjugation of the verb كَتَبَ *kataba* (to write). By inserting prefixes and suffixes to the root كتب *ktb* ’ktb’ we get all the other forms such as in Table 2.1.

  So the verb كتب *ktb* in the past has 14 different forms whereas if we consider an English example: The verb *write* in the past has only one form:

  - I wrote (First singular)
  - You wrote (Second singular and plural)
  - He/She wrote (Third singular)
  - We wrote (First plural)
  - They wrote (Third plural)

- **The presence of many weak verbs:** Roots containing one or two of the long vowels و *w*, ي *y*, ا *ā* or أ ’”hamza” often lead to verbs with special phonological rules because these radicals can be influenced by their surroundings. Such verbs are called *weak verbs*. These special rules are according to the position of the long vowel in the root. Normal verbs where all letters of the root are consonants, when conjugated or preceded by a negation, they follow certain pattern rules. However, weak verbs when conjugated or preceded by negation will result in a change of the written form, such as a deletion of a letter. For example the written form يعد *yʕd* which its infinitive is عَاد *ʕād*. The middle letter is a long vowel, and therefore it can have several interpretations as follows:

| Pronoun | Past | Present |
|---|---|---|
| أَنَا *ʾanā* | كَتَبْتُ *katabtu* | أَكْتُبُ *ʾaktubu* |
| نَحْنُ *naḥnu* | كَتَبْنَا *katabnā* | نَكْتُبُ *naktubu* |
| أَنْتَ *ʾanta* | كَتَبْتَ *katabta* | تَكْتُبُ *taktubu* |
| أَنْتِ *ʾanti* | كَتَبْتِ *katabti* | تَكْتُبِينَ *taktubyna* |
| أَنْتُمَا *ʾantumā* | كَتَبْتُمَا *katabtumā* | تَكْتُبَانِ *taktubāni* |
| أَنْتُمَا *ʾantumā* | كَتَبْتُمَا *katabtumā* | تَكْتُبَانِ *taktubāni* |
| أَنْتُمْ *ʾantum* | كَتَبْتُمْ *katabtum* | تَكْتُبونَ *taktubwna* |
| أَنْتُنَّ *ʾantunna* | كَتَبْتُنَّ *katabtunna* | تَكْتُبْنَ *taktubna* |
| هُوَ *huwa* | كَتَبَ *kataba* | يَكْتُبُ *yaktubu* |
| هِيَ *hiya* | كَتَبَتْ *katabat* | تَكْتُبُ *taktubu* |
| هُمَا *humā* | كَتَبَا *katabā* | يَكْتُبَانِ *yaktubāni* |
| هُمَا *humā* | كَتَبَتَا *katabatā* | تَكْتُبَانِ *taktubāni* |
| هُمْ *hum* | كَتَبُوا *katabuwā* | يَكْتُبونَ *yaktubwna* |
| هُنَّ *hunna* | كَتَبْنَ *katabna* | يَكْتُبْنَ *yaktubna* |

Table 2.1: The conjugation of the verb كَتَبَ *kataba* (to write) in the past and present tenses

  - عَاد *ʿād* (returned) → لم يَعُد *lm yaʿud* (did not return)
  - أَعَاد *ʾaʿād* (Brought back) → لم يُعِد *lm yuʿid* (did not bring back)
  - أَعدّ *ʾaʿdd* (prepared) → لم يُعِدّ *lm yuʿidd* (did not prepare)
  - عَدّ *ʿadd* (counted) → لم يَعُدّ *lm yaʿudd* (did not count)
  - وعد *wʿd* (promised) → لم يَعِد *lm yaʿid* (did not promise)

- **The nunation:** Nunation diacritics means doubling the short vowel and can only occur in word final positions in nouns, adjectives and adverbs, where they describe indefiniteness. They sound like the short vowel followed by an unwritten ”n” sound. For instance كِتَابٌ *kitābun* (The book) where the written form in MSA is كتَاب *ktāb*.

- **Shadda:** If nunation is doubling the vowel, shadda is doubling the consonant. Similarly, the ambiguity comes from the fact that this doubling is not explicit in the written form. Many MSA words have one of the sounds doubled which is not written. For example: علم *ʿlm* can be interpreted as عَلَّمَ *ʿllama* which means *teach*, or عَلِمَ *ʿalima* which means *knew*.

- **different affixes have the same written form:** for instance: the prefix ت *t* can indicate third person singular feminine in هي تكتب *hy tktb* (she writes), or second person singular masculine in أَنت تكتب *ʾant tktb* (you write).

- **Is it an affix or part of the main root:** this comes from the long vowels being part of the main root, or they are affixes, such as: the letter أَ *ʾa* in أَسَد *ʾasad* (meaning lion) where it is part of the main root and أَسُدّ *ʾasudd* meaning (I block), where it is a prefix of the verb سَدَّ *sadda*.

- **Accusative form:** When it is accusative the dual is written in the same way as the plural: اللَّاعبَيْن *āl-lāʿbayn* (two players) اللَّاعبِين *āl-lāʿbiyn* (the players).

- **the letter ي *y*:** The letter ي *y* can play two roles, one possessive which corresponds to ”*my* ” in English, and the other one corresponds to ”*ish*” in English, When adding it to any noun it produces an adjective related to that noun. For example:

- كِتَاب *kitāb* + ي *y* → كِتَابِي *kitāby* → My book
- كِتَاب *kitāb* + ي *y* → كِتَابِيّ *kitābiyy* → people who follow the book

#### 2.4.2.4 Syntactic ambiguity

Although all natural languages share the fact they are all syntactically ambiguous, it is the linguistic features of the language that contribute towards increasing or decreasing the level of ambiguity. We consider MSA as a highly syntactically ambiguous due to many reasons which are described below:

- **Free word order:**

  Arabic is a free word order language, where the subject, object and complement can appear anywhere around the verb without affecting the structure of the sentence. This results in more syntactic ambiguity, and therefore requires more complex analysis.

  The normal word order for Arabic is VSO. I.e. the first item is the verb, then the subject, and finally the object. However the other forms are also possible: SVO, VOS and OVS. It is not easy for the parser to know which order is for a given sentence. This is mainly due to the lack of diacritics, because the distinction between a nominative subject and an accusative object is known through the last diacritic of the word, and this is missing in MSA.

  Depending on whether the first word is a verb or a noun, two types of sentences are mentioned here: Nominal, and Verbal sentences. I.e. nominal sentences are headed by a noun and verbal sentences are headed by a verb. For a simple verbal sentence that contains three elements: verb, subject and object, all the different word orders SVO, VSO, VOS, and OVS are possible.
  An example of these combinations is highlighted in Table 2.2:

| Word order | Example | Translation |
|---|---|---|
| VSO | كتب الولد الدرس *ktb ālwld āldrs* | The boy has written the lesson |
| SVO | الولد كتب الدرس *ālwld ktb āldrs* | |
| VOS | كتب الدرس الولد *ktb āldrs ālwld* | |
| OVS | الدرس كتب الولد *āldrs ktb ālwld* | |

Table 2.2: Arabic free word order

- **Agreement:** What helps us decide the right word order is the case and agreement features. The agreement features that can be considered here are: number (singular, plural, dual), gender (masculine, feminine), and person (1st, 2nd, 3rd). The case features include: nominative, accusative and genitive. In VSO order, the verb agrees with the subject only in gender, and the verb is marked in singular, whether the subject is singular, dual, or plural.

  1. كتب الولد الدرس *ktb ālwld āldrs* (the boy wrote the lesson)
  2. كتب الولدَان الدرس *ktb ālwldān āldrs* (the two boys wrote the lesson)
  3. كتب الأَولَاد الدرس *ktb āl؞awlād āldrs* (the boys wrote the lesson)
  4. كتب البنت الدرس *ktb ālbnt āldrs* (the girl wrote the lesson)

  1, 2, 3 are allowed sentences in Arabic, however 4 is not acceptable because the verb is masculine and the subject is feminine.

  In SVO order the verb agrees with the subject in gender and number. For example:

  1. الولد كتب الدرس *ālwld ktb āldrs* (the boy wrote the lesson)
  2. الولدَان كتبَا الدرس *ālwldān ktbā āldrs* (the two boys wrote the lesson)
  3. الأَولَاد كتبُوا الدرس *āl؞awlād ktbwuā āldrs* (the boys wrote the lesson)

  There various types of agreement relationships as shown in the examples below:

  – The demonstrative pronoun in MSA agrees with the noun in number and gender such as :هذَا الرجل *hḏā ālrğl*, هذَان الرجلَان *hḏān ālrğlān* and هَاته المرأَة *hāth ālmr؞ah* which respectively corresponds to "*this man*, *these two men* and *this woman*".
  – Adjectives agree with nouns in number, gender, case and definiteness: رأَيت الطَالبَين النَاجحَين *r؞ayt ālṭālbayn ālnāğḥayn* (I saw the two successful students).

- Relative pronouns agree with nouns in number, gender and case: رأَيت الطَالبَتَين اللّتَين نجحَتَا في الإمتحَان *rʾayt ālṭālbatayn āl-ltayn nğḥatā fy āl-ʾimtḥān* (I saw the two female students who passed their exam).
- Unlike the VSO order where the verb only agrees in gender and person (the verb is in the default singular form), in the SVO order the subject agrees with the verb in number, gender and person: for example *The girls wrote the lesson*:
  - VSO order: كتبت البنَات الدرس *ktbt ālbnāt āldrs*
  - SVO order: البنَات كتبن الدرس *ālbnāt ktbn āldrs*

**Nominal sentences** In addition to verbal sentences, MSA allows for nominal sentences, which are sometimes referred to as copular constructions or equational sentences [Hab10]. They consist of an *NP* and a predication (such as another *NP*, an adjective or a *PP* ). In its simplest form the subject is a definite noun, proper noun or a pronoun in the nominal case and the predicate is an indefinite nominal noun, proper noun or an adjective that agrees with the subject in number and gender. For example:

- الكتَابُ مفيدٌ *ālktābu mfydun* (The book is new): Both nouns are singular masculine.
- الطَّالبَان غَائبَان *ālṭṭālbān ġāʾibān* (The two students are absent): Both nouns are dual masculine.
- هو في البيتِ *hw fy ālbyti* (He is in the house): the predicate is a *PP*.

In the current parser, these nominal sentences are described by using a rule in a Categorial approach that says: *NP* can be considered as sentence missing a predicate.

**Construct NPs** Another aspect of phrase structure in MSA is that nouns can combine with *NPs* to form ”construct *NPs*” mainly to express possession. One example of these complex structures: an indefinite noun is followed by a definite noun. These constructions are known as الإِضَافة *āl-ʾiḍāfh*. This is roughly similar to the possession construction and noun compounds in English. Arabic is unlike languages like English and French in that there is not explicitly

a word for "*of*" in the possessive sense. For example كتب الولد *ktb ālwld* which might mean "the books of the boy". Because of this, one of the current parses had the first two nouns as an NP.
The following example combine 3 NPs: صوَاريخ بعيدة المدَى *ṣwāryḫ bʿydh ālmdā* (long range missiles). The last NP or the one who possesses the object has to be definite, i.e it starts with ال *āl*, whereas the preceding *NPs* have to be indefinite (without ال *āl*). The last NP does not require ال *āl* if it is a proper noun, or a pronoun. For example: وِلَاية فلوريدَا *wlāyh flwrydā* (The state of Florida). combining *NPs* with *NPs* to make an *NP* makes MSA a very ambiguous language, because any sequence of two words where there is a possibility of them being nouns, the parser will combine them together to make an *NP* resulting in lots of unnecessary parses. Things get even more complicated because it does not stop at two NPs combined together. This can be extended recursively by combining several NPs to make one resulting NP. for example:

- ابن عم رَئِيس مجلس نوَاب برلمَان كوريَا الجنوبية *ābn ʿm raʾys mǧls nwāb brlmān kwryā ālǧnwbyh* 'The son of the uncle of the president of the Council of the South Korean parliament'

- **Zero subject or pro drop nature of MSA [Att08]** Pro drop feature in any language means that a null category (pro) can be introduced in the subject position. Similar to other languages, in MSA, the subject pronoun can be omitted. Not only the missing subject by itself is a problem, but the combination of this and the other features of MSA. For example, Arabic verbs are either transitive or intransitive. Therefore, It is not clear whether a sequence of a verb followed by a noun is an intransitive verb and a subject, or a transitive verb and an object, where the subject is missing. By taking the active nd the passive form into consideration, things get more complicated, since the same sequence of a verb followed by a noun can be interpreted as a passive verb and its subject. According to [Att08] there are two challenges: One is to decide whether a pro drop or not, and the other one is recovering the pronoun reference after deciding that there is a pro drop.

It is worth noting that every feature described in this chapter can contribute to the ambiguity of MSA. However, what makes MSA extremely ambiguous is the combination of these features together.

# Chapter 3

# Background: POS Tagging

Several techniques have been used for building POS taggers which fall into three main categories:

- Rule based taggers [BBG71, Vou95].
- Stochastic taggers [STC65, BM76, Mar87, Gar87, Chu88, **?**].
- Transformation based taggers [Bri95].

One of the main distinctions that one can see among POS taggers is in terms of the degree of automation of the training and tagging process. Here we use the term supervised vs unsupervised.

## 3.1 Supervised vs Unsupervised

Supervised taggers rely on pre-tagged corpora which are used for training, and extracting information such as: the lexicon, word/tag frequencies, and set of rules, etc. The larger is the size of the corpora, the better is the performance [HUK07]. The disadvantage of supervised tagging is that pre-tagged corpora are not readily available for many languages which one might want to tag.

On the other hand, unsupervised taggers do not require a pre-tagged corpus, but instead use sophisticated computational methods to automatically induce tagsets, and based on this information, they can calculate the probabilistic information needed by stochastic taggers, or induce the rules needed by rule-based systems. The main benefit of using a fully automated approach to POS tagging is that it is extremely portable, and can be applied to any corpus of text. Unsupervised approaches address the need

to accurately tag previously untagged languages, because hand tagging is a costly and very time consuming process. There are, however, drawbacks to the unsupervised approach. The word clustering devised by automatic taggers tends to be very coarse, i.e. one loses the fine distinctions found in the carefully designed tag sets used in the supervised methods [Gui95].

Both supervised and unsupervised models can be of the following types: Rule-based approach and stochastic approach. Early approaches to part-of-speech tagging were rule-based ones. After 1980's, statistical methods became more popular. More recently, all of the approaches are used together to get better results.

## 3.2 Rule-based approach

Rule-based part-of-speech tagging is a method that uses manually written rules for tagging. It is based on a dictionary or a lexicon to get possible tags for each word to be tagged. The rules are used to identify the correct tag when a word has more than one possible tag. Analysing the linguistic features of the word, the surrounding of the word and other aspects will help assigning the most appropriate tag. This approach consists of two stages. First stage assigns each word a list of potential parts of speech by using a dictionary. Second stage uses hand written disambiguation rules to cut down the list to a single part of speech for each word. The rule-based approach uses contextual and morphological rules to assign POS tags to words. These rules are often called context frame rules. A context frame rule is designed by a grammarian by observing a set of data which give some information about a potential tag in the context of up to three tags on either side. In addition the rule could specify the potential tag was impossible in this context so that one of the other potential tags must be the correct one [JM00]. For instance, an English rule might say: If an unknown word is preceded by a determiner and followed by a noun, then tag it as an adjective. These rules can either be handwritten, or automatically induced by the tagger after the training. Manually writing the rules is a repeated process over a long period because it requires analysing the errors made by the tagger, and carefully refining the rules. Therefore, this is considered as extremely time consuming, and difficult task to do. The first rule-based tagger was done by Greene and Rubin in 1970 when they tagged the Brown corpus [BBG71]. They used a rule-based approach to build their tagger which was called TAGGIT. It was a set of rules to choose the appropriate tag for each word in a given context. This

tagger achieved an accuracy of 77%, and the remainder was later tagged manually. The idea at the time was to provide a corpus annotated with part of speech information as a useful tool for linguistic researchers. They used the Brown corpus which had one million words of written American English collected in the early 1960's, and a tagset consisting of 77 tags. The idea was to assign each word with a set of possible tags, and then analyse the surrounding to decide the correct one.

Another successful rule-based approach has also been used by Karlsson [Kar90] and Koskenniemi [Kos90] in the form of a constraint grammar. In this framework, the problem of parsing was broken into seven sub problems or modules:

- Preprocessing
- Morphological analysis
- Local morphological disambiguation
- Morpho-syntactic mapping
- Context dependant morphological disambiguation
- Determination of intrasentential clause boundaries
- Disambiguation of Surface syntactic functions

The first four of these modules are related to morphological disambiguation, and the rest are used for parsing the running text. This work has achieved a slightly more than 90% success rate of words with unique syntactic labels. The rest have more than one syntactical label, one of which is correct. Rule-based taggers usually require supervised training, but also a good amount of work has been done in automatic induction of rules. One way of doing this is to run an un-tagged text through a tagger, and investigate its performance. One can then go through the output, and correct any incorrectly tagged words by hand. This tagged text is then run through the tagger, which learns correction rules by comparing the two sets of data. Repeating these steps will make the tagger achieve higher performance. One example of this is Brill's tagger which achieved an accuracy of 96%. Nearly all rule-based taggers after 1992 give reference to Brill's work [Bri92]. The drawback of this approach is that it is difficult to write all the rules required for this task. It is also a very time consuming approach.

## 3.3 Stochastic approach

Stochastic approach or sometimes it is called Statistical approach refers to any approach that somehow includes frequency, probability, or statistics. The use of probabilities in tagging is an old idea, since it was first introduced by [STC65]. A complete probabilistic tagger was introduced by Bahl and Mercer [BM76]. Then later in the 1980s several stochastic taggers were introduced [Mar87, Gar87, Chu88, **?**].

A large corpora is used to get statistical information. First, a corpus can be divided into two parts: one part is used for the training phase, to get a statistical model, which will be used to tag untagged texts. The other part is used to test the statistical model. A very simple stochastic tagger can disambiguate words based solely on the probability that this word occurs with a specific tag. I.e. the tag found most in the training data, will be assigned to an ambiguous instance of that word. One disadvantage of this is that, while it may assign a correct tag to a given word, it may also give an incorrect tag. A better approach than the word frequency is the N-gram approach. N-gram of size 1 is called a unigram, size 2 is called bigram, size 3 is called trigram, and size n is N-gram. The idea here is that, if we have a sequence of words, each with one or more potential tags, then we can choose the most likely sequence of tags by calculating the probability of all possible sequences of tags, and then choosing the sequence with the highest probability. This approach calculates the probability of a given sequence of tags occurring. The most likely tag for a given word is determined by the probability that it occurs with N previous tags. That is choosing a preferred tag for a word by calculating the most likely tag in the context of the word and its immediate neighbours. The best known algorithm for implementing an n-gram is the Viterbi algorithm, which is a dynamic programming technique. It is an algorithm to compute the most likely state sequence in a hidden Markov model given a sequence of observed outputs. By using the Hidden Markov Model we can directly observe the sequence of words, but we can only estimate the sequence of tags, which is ‘hidden’ from the observer of the text; hence the term ”hidden Markov model” is appropriate. A HMM enables us to estimate the most likely sequence of tags, making use of observed frequencies of words and tags in a training corpus. One of the earliest probabilistic taggers was called CLAWS (Constituent Likelihood Automatic Word-tagging System), developed at the University of Lancaster [LGB94].

## 3.4 Transformation Based Tagging

This is sometimes called Brill tagging and is based on Transformation Based Learning (TBL) approach to machine learning. Brill [Bri95] combined advantages of both rule based and stochastic taggers. TBL is based on rules that specify what tags are assigned to what words like the rule based tagger, and is a machine learning technique in which rules are automatically induced from data like the stochastic taggers. It uses a supervised learning technique which assumes a pre-tagged training corpus. An example of TBL is:

- Sentence 1: John is expected to race tomorrow.
- Sentence 2: John won the race.

In sentence 1 the word ”race” functions as a verb, whereas in sentence 2 it functions as a noun. The Brill tagger labels every word with its most likely tag from a tagged corpus. From the Brown corpus, ”race” is most likely to be a noun. Therefore the tagger will label both instances of the word ”race” as a noun (NN). This is correct for sentence 2 but not correct for sentence 1 where race is a verb (VB). Here comes the transformation rules which replace the tag NN with VB for all instances of ”race” that are preceded by tag TO.
TBL was first introduced by Eric Brill in 1994 and achieved an accuracy of 97.2%. Figure 3.1 shows how TBL works. Various initial state annotators can be used, including: the output of a stochastic tagger or labeling all words with their most likely tag. By comparing the results of the initial state with a gold-standard corpus, an ordered list of transformations can be learnt, which can be applied to the output of the initial state annotator to improve its results. This learning continues until no transformation (whose application improves the annotated corpus) can be found.

## 3.5 Arabic and Part of Speech tagging

The linguistic characteristics of Arabic present some interesting challenges to NLP researchers. For languages like English, word-level POS tagging seems sufficient since words usually correspond to the syntactically relevant POS tag classes [BhW07]. However, for Arabic, the syntactically relevant POS tag classes do not necessarily correspond to words. Arabic words are often formed by concatenating smaller parts, each of which has its own POS tag. The main part of the word is the stem which is

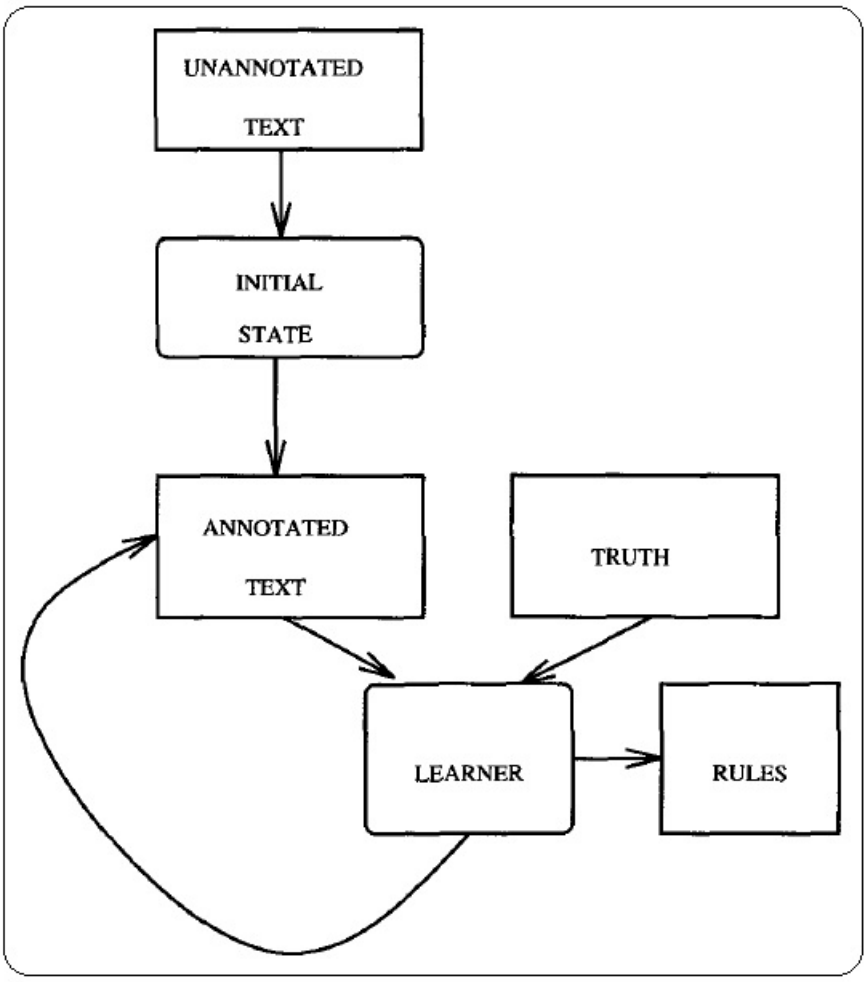



Figure 3.1: Transformation Based Learning (TBL) approach [Bri95]

often derived from a root morpheme. The other parts are affixes and clitics.

Clitics include certain prepositions, conjunctions, determiners and pronouns. The affixes include inflectional markers for tense, gender, and number. In Arabic before Part of Speech tagging, the process of tokenization must be executed, which is the process of separating clitics from stems.

### 3.5.1 Part of speech tagsets

Traditional Arabic grammarians used to break down the words into three categories:

- **Noun الاسم** *ālāsm*: Nouns are names of people, animal and things. A noun can either be masculine or feminine. Nouns can be definite such as **الكتاب** *ālktāb* 'the book' and **القلم** *ālqlm* 'the pen' or indefinite such as **ولد** *wld* 'a boy' and **فيل** *fyl* 'an elephant'. The following are also considered to be nouns:
  - **Subject pronouns:** such as **أنَا** *ʔnā* 'I', **أنت** *ʔnt* 'you', **هو** *hw* 'he' and **هي** *hy* 'she'.
  - **Object pronouns:** such as **ه** *h* 'him' and **هَا** *hā* 'her'.
  - **Demonstrative pronouns:** for example: **هذَا** *hḏā* 'this' and **ذلك** *ḏlk* 'that'.
  - **Possessive pronouns:** for example: **كتَابه** *ktābh* 'my book' and **كتَابهَا** *ktābhā* 'her book'.
  - **relative pronouns**: for example: **هذه السيَاره التي اردت ان اشتري** *hḏh ālsyārh ālty ārdt ān āštry* 'This is the car that I want to buy.
  - **Adjectives:** In English adjectives come before the noun they describe. However, in Arabic, adjectives follow the noun they describe and they agree in gender and number.
- **Verb الفعل** *ālfʕl*:
  - Verbs in Arabic can be transitive or intransitive: Like **رَاجع الولد درسه** *rāǧʿālwld drsh* 'The boy revised his lesson' and **نَامت البنت** *nāmt ālbnt* 'The girl slept'.

- **Voice:** Like English, there are two voices: active and passive. أَكَلَ الولد برتقَالة *ʾakala ālwld brtqālh* 'the boy ate an orange', and أُكِلَت البرتقَالة *ʾukilat ālbrtqālh* 'the orange was eaten'.
- **Tense:** A tense indicates the time of the verb action. In Arabic, there is only two tenses: past (perfect) and present (imperfect). for example: كتب الولد الدرس *ktb ālwld āldrs* 'The boy wrote the lesson' and يكتب الولد الدرس *yktb ālwld āldrs* 'The boy is writing the lesson'.
- **Mood:** The mood indicates the writer's or speaker's attitude toward what he is writing or saying. In Arabic there are three types of mood: indicative, imperative, and subjunctive mood. The indicative mood is used to make a statement like استقَال الوزير من منصبه *āstqāl ālwzyr mn mnṣbh* 'the minister resigned from his post'. The imperative mood is used to give instructions, orders or requests. The subjunctive mood is used in English much less than in many other languages. The subjunctive mood is the verb form used to express a wish, a suggestion, a command, or a condition that is contrary to fact. The form of a verb in the subjunctive mood may be different from the form with the same subject which is not in the subjunctive mood. An English example is as follows:
  - I was in your position two years ago. (Indicative mood)
  - If I were in your position, I would do the same. (Subjunctive mood)

- **Particle الحرف** *ālḥrf*: Particles in Arabic consist of short words such as:
  - **Prepositions:** such as من *mn* 'from', في *fy* 'in', بِ *bi* 'with'.
  - **Subjunctive particles:** such as أَن *ʾan* and لَن *lan*.
  - **Jussive particles:** such as لم *lm* and لمَّا *lmmā*.
  - **Negative particles:** such as مَا *mā* and لَا *lā*.

Habash considered this traditional classification to be very coarse and therefore cannot be used computationally [Hab10]. Because of the highly complex inflected and derivational morphology, Arabic POS tagsets can be very large. In this section, we present some known Arabic tagsets in the field.

#### 3.5.1.1 The Buckwalter tagset

The Buckwalter tagset was developed by Tim Buckwalter. It is a form-based tagset that can be used for tokenized and untokenized text [Hab10]. The untokenized tags are the output of Buckwalter Arabic Morphological Analyzer [Buc04], and the tokenized tags are used in the Penn Arabic Treebank PATB [MB04]. This tagset uses around 70 subtag symbols such as (DET 'determiner', NSUFF 'Nominal Suffix', and ADJ 'Adjective) as in Table 3.1 . These subtags are combined together to form over 170 morpheme tags, such as (NSUFF_FEM_SG 'feminine singular nominal suffix' and CASE_DEF_ACC 'accusative definite'. Similarly word tags are produced by concatenating morpheme tags together. for example the Arabic word الذكية *āl*ḏ*kyh* 'the smart' can be tagged as DET+ADJNSUFF_FEM_SG+CASE_DEF_ACC. Whereas a Buckwalter untokenized tagset can have thousands of tags, the tokenized one can have around 500 tags.

#### 3.5.1.2 Reduced Buckwalter tagset

Reduced tagset RTS which is also called The Bies tagset, was developed by Ann Bies and Dan Bikel. This tagset reflects number for nouns and some tense information for verbs. However, gender, definiteness, and case information are lost for nouns and verbs, number information is lost for adjectives and mood, future tense, and aspect information are lost for verbs. RTS consists of 24 POS tags reduced from the 139 tags created by the Buckwalter Arabic Morphological Analyzer [SvdBN07]. This was created to enhance compatibility between the English and Arabic treebanks. This tagset is considered to be very coarse since it ignores many distinctions that we have seen in the full Buckwalter tagset. For example: JJ is used for all adjectives regardless of their inflections, whereas in the full Buckwalter tagset there are different tags for an adjective like (ADJ, ADJ_COMP, ADJ_NUM, ...etc). The tags in this reduced set are:

- **Nominals:**
  - Nouns: NN *singular common noun*, NNS *plural or dual common noun*, NNP *singular proper noun*, NNPS *plural dual proper noun*.
  - Pronouns: PRP *personal pronoun*, WP
  - Other: JJ *adjective*, RB *adverb*, WRB *relative adverb*, CD *cardinal number*, FW *foreign word*.

| Tag | Meaning | Tag | Meaning |
|---|---|---|---|
| VERB | verb | NOUN | noun |
| PSEUDO_VERB | pseudo-verb | NOUN_NUM | nominal/cardinal number |
| PV | perfective verb | NOUN_QUANT | quantifier noun |
| PV_PASS | perfective passive verb | NOUN_VN | deverbal noun |
| PVSUFF_DO:<PGN> | direct object of perfective verb | NOUN_PROP | proper noun |
| PVSUFF_SUBJ:<PGN> | subject of perfective verb | ADJ | adjective |
| IV | imperfective verb | ADJ_COMP | comparative adjective |
| IV_PASS | imperfective passive verb | ADJ_NUM | adjectival/ordinal number |
| IVSUFF_DO:<PGN> | direct object of imperfective verb | ADJ_VN | deverbal adjective |
| IV<PGN> | imperfective verb prefix | ADJ_DROP | proper adjective |
| IVSUFF_SUBJ:<PGN> | imperfective verb subject | ADV | adverb |
| MOOD:<Mood> | mood suffix | REL_ADV | relative adverb |
| CV | imperative verb | INTERROG_ADV | interrogative adverb |
| CVSUFF_DO:<PGN> | imperative verb object | PRON | pronoun |
| CVSUFF_SUBJ:<PGN> | imperative verb subject | PRON_<PGN> | personal pronoun |
| PREP | preposition | POS_PRON_<PGN> | possessive personal pron |
| CONJ | conjunction | DEM_PRON_<GN> | demonstrative pronoun |
| SUB_CONJ | subordinating conjunction | REL_PRON | relative pronoun |
| PART | particle | INTERROG_PRON | interrogative pronoun |
| CONNEC_PART | connective particle | NSUFF<Gen><Num><Cas><Str> | nominal suffix |
| EMPHATIC_PART | emphatic particle | CASE<Def><Cas> | nominal suffix |
| FOCUS_PART | focus particle | DET | determiner |
| FUT_PART | future particle | PUNC | punctuation |
| INTERROG_PART | interrogative particle | ABBREV | abbreviation |
| JUS_PART | jussive particle | INTERJ | interjection |
| NEG_PART | negative particle | LATIN | latin script |
| RC_PART | response conditional particle | FOREIGN | foreign word |
| RESTRIC_PART | restrictive particle | TYPO | typographical error |
| VERB_PART | verb particle | PARTIAL | partial word |
| VOC_PART | vocative particle | DIALECT | dialect word |

Table 3.1: The Buckwalter tagset components

- **Particles:** CC *coordinating conjunction*, DT *determiner or demonstrative pronoun*, RP *particle*, IN *preposition or subordinating conjunction*.
- **Verbs:** VBP *active imperfect verb*, VBN *passive imperfect or perfect verb*, VBD *active perfect verb*, VB *imperative verb*.
- **Others:** UH *interjection*, PUNC *punctuation*, NO_FUNC *unanalyzed word*.

#### 3.5.1.3 The CATiB POS tagset

This tagset was developed for the Columbia Arabic Treebank project CATiB [HR09a]. The idea of creating this tagset was inspired by the traditional classification of words and therefore was made up very simple. The point of making this way is to speed up the human annotation process. The tagset only consists of 6 POS tags:

- **VRB:** This represents all types of verbs even incomplete verbs.
- **VRB_PASS:** This is for passive voice verbs.
- **NOM:** This is for all nominals including: nouns, adjectives, adverbs, active or passive participle, deverbal noun مصدر *mṣdr*, pronouns (relative, interrogative, personal, demonstrative, interrogative), numbers and interjections.
- **PROP:** This is used for proper nouns.
- **PRT:** This is for all particles including prepositions, conjunctions, negative particles, and definite article.
- **PNX:** for all punctuation marks.

An automatically extended version of CATiB was created. These extensions increased the tagset size to 44 tags. It attached the prefix or suffix to the noun for example العلمون *ālmʕlmwn* 'The teachers' which is tagged as NOM, will be tagged in the extended version as Al+NOM+wn.

#### 3.5.1.4 The KHOJA tagset

They derived their own tagset rather than using a European based tagset. Therefore, they used the traditional three main parts of speech of Arabic: Noun, Verb, and Particle [SKK01]. An Arabic noun is a person, place or thing. Proper nouns are the names of

people, places or things. The noun class also includes what in the traditional grammatical theory would be classified as: participles, pronouns, relatives, demonstratives, and interrogatives. A verb class in Arabic is similar to that of European languages. However, the tenses and aspects are different. The verb class includes the subcategories: perfect, imperfect and imperative. An Arabic particle is a preposition, adverb, conjunction, interrogative particles, exceptions and interjections. It was reported that the tagset contains 177 tags: 103 nouns, 57 verbs, 9 particles, 7 residuals, and 1 punctuation. They are made by joining specific sequences followed by the relevant attributes. That is, every base tag such as noun or verb has specific attributes that can be attached to it. A perfective verb (past) for instance can have attributes like number, person, gender, whereas the imperative verb can only have two attributes: number and gender in addition to the second person. An example of this sequence is : VIS3F (third-person singular feminine imperfect verb). The following is the khoja tagset which shows the POS and the attributes relevant to them.

- **N** noun
  - **C** common + **Attributes:** number, gender, case, definiteness.
  - **P** proper. It has no attributes
  - **Pr** pronoun. There are three types:
    - **P** personal + **Attributes:** number, person, gender.
    - **R** relative
      - **S** specific + **Attributes:** number, gender.
      - **C** common
    - **D** demonstrative + **Attributes:** number, gender.
  - **Nu** numerical
    - **Ca** cardinal + **Attribute:** [Sg]-gender.
    - **O** ordinal + **Attribute:** [Sg]-gender.
    - **Na** numerical adjective + **Attribute:** [Sg]-gender.
  - **A** adjecive + **Attributes:** number, gender, case, definiteness.
- **V** verb
  - **P** perfective + **Attribute:** number-person-gender.
  - **I** imperfective + **Attribute:** number-person-gender-mood.

  - **PV** imperative + **Attribute:** number-[2]-gender.

- **P** particle
  - **Pr** preposition, **A** adverbial, **C** conjuncion, **I** interjection, **E** exception, **N** negative, **A** answers, **X** explanations, **S** subordinates.
- **R** residual
  - **F** foreign, **M** mathematical, **N** number, **D** day of the week, **MY** month of the year, **A** abbreviation, **O** other.
- **PU**punctuation.
- Attributes:
  - Gender: **M** masculine, **F** feminine, **N** neutral.
  - Number: **S** singular, **Du** dual, **Pl** plural.
  - Person: **1** first, **2** second, **3** third.
  - Case: **N** nominative, **A** accusative, **G** genitive.
  - Definiteness: **D** definite, **I** indefinite.
  - Mood: **I**indicative, **S** subjective, **J**jussive.

#### 3.5.1.5 The PADT tagset

This tagset was developed for the Prague Arabic Dependency Treebank [HSZ+04]. Similar to the Khoja tagset, tags consist of two parts: POS and Features. The POS are illustrated here:

- Verbs:
  - **VI** :imperfect verb.
  - **VP** :perfect verb.
  - **VC** :imperative verb.
- Nouns:
  - **N** :noun.
  - **A** :adjective.

  - **D** :adverb.
  - **Z** :proper noun.
  - **Y** :abbreviation

- Pronouns:
  - **S** :pronoun.
  - **SD** :demonstrative pronoun.
  - **SR** :relative pronoun.

- Particles:
  - **F** :particle.
  - **FI** :interrogative particle.
  - **FN** :negative particle.
  - **C** :conjunction.
  - **P** :prepositioon.
  - **I** :interjection.

- Other:
  - **G** :graphical symbol.
  - **N** :number.

Then the second part (the features) consists of a 7 character string. Each character relates to the relevant attribute:

1. **Mood: I** indicative, **S** subjunctive, **J** jussive.
2. **Voice: A** active or **P** passive.
3. **Person: 1** (I, We), **2** (You), **3** (He,She,they).
4. **Gender: M** masculine, **F** feminine.
5. **Number: S** singular, **D** dual, **P** plural.
6. **Case: 1** nominative, **2** genitive, **3** accusative.
7. **Definiteness: I** indefinite, **D** definite, **R** reduced, **C** complex.

### 3.5.2 Current Arabic morphological analyzers

In here, we described some of NLP tools that were developed to process Modern Standard Arabic. We describe BAMA which is the Buckwalter Arabic Morphological Analyzer. Then, The Xerox Arabic morphological analyzer which was produced in 1996 at the Xerox Research Centre Europe. ALMORGEANA and MAGEAD are morphological analysis and generation systems. MADA is for morphological analysis and disambiguation including POS tagging, lemmatization and diacriticization. TOKAN is a general tokenizer that works with MADA.

#### 3.5.2.1 The Buckwalter Arabic Morphological Analyzer

The first approach to Arabic tokenization and POS tagging is the one adopted in the Arabic Treebank, which relies on manually choosing the appropriate analysis from among the multiple analyses given by AraMorph (an Arabic rule-based morphological analyzer and part-of-speech tagger by Buckwalter). The Buckwalter Arabic Morphological Analyzer [Buc04] is becoming the most respected lexical resource of its kind. It has been used in the Penn Arabic Treebank, and the Prague Arabic Dependency Treebank. It uses a concatenative lexicon-driven approach in which morphotactics and orthographic rules are built directly into the lexicon itself instead of being specified in terms of general rules that interact to realise the output. The system is composed of three elements: the lexicon, the compatibility tables, and the analysis engine.

- **Lexicon:** Words are composed of three elements: prefix, stem, and suffix, where the prefix is limited to 0-4 characters, and the suffix can have 0-6 characters. So the prefix and suffix can be null. The lexicon contains three files:
  - dictPrefixes contains all Arabic prefixes and their concatenations.
  - dictSuffixes contains all Arabic suffixes and their concatenations.
  - contains all Arabic stems.

  All possible concatenations of prefixes and suffixes will be specified in the lexicon entries. Every lexicon entry determines a morphological compatibility category, an English gloss, and a POS tag as shown in Table 3.2. The English glosses will allow this lexicon to function as a dictionary.

- **Compatibility tables:** Compatibility tables should show information on which morphological categories are allowed to concatenate with other categories such

as noun or verb categories. For example the morphological category for the prefix conjunction "w" (and) is compatible with all noun categories and perfect verb categories. However, the same conjunction "w" is not compatible with imperfect verb categories because imperfect verb stems are preceded with a subject prefix. There are three compatibility tables, each of which lists pairs of compatible morphological categories. The first table lists compatible Prefix and Stem morphological categories. The second one lists compatible Prefix and Suffix morphological categories, and the third one lists compatible Stem and Suffix morphological categories as shown in Table 3.3.

- **Analysis engine:** the lexicon and the compatibility contribute to most of the hard decisions. Analysis engine provides a simple algorithm to help make the necessary decisions by producing multiple analyses as tuples of: full diacritization, lemma, morpheme analysis and POS tags.

| | | | | | |
|---|---|---|---|---|---|
| و *w* /wa | Pref-Wa | *and* | ;; 1_كتب *ktb* /katab-u_1 | | |
| ب *b-* /bi | NPref-Bi | *by/with* | كتب *ktb* /katab | PV | *write* |
| وب *wb-* /wabi | NPref-Bi | *and+by/with* | كتب *ktb* /kotub | IV | *write* |
| ال *ā-* /Al | NPref-Al | *the* | كتب *ktb* /kutib | PV_pass | *be written* |
| بال *biā-* /biAl | NPref-BiAl | *with/by+the* | كتب *ktb* /kotab | IV_Pass_yu | *be written* |
| وبال *wbiā-* wabiAl | NPref-BiAl | *and+with/by+the* | ;; 1_كتاب *ktāb* /kitAb_1 | | |
| اة *-ah* /ap | NSuff-ap | *[fem-sg]* | كتاب *ktāb* /kitAb | Ndu | *book* |
| اتان *-atān* /atAni | NSuff-atAn | *two* | كتب *ktb* /kutub | N | *books* |
| اتين *-atayn* /atayoni | NSuff-tayn | *two* | ;; 1_كتابة *ktābh* /kitAbap_1 | | |
| اتاه *-atāh* /atAhu | NSuff-atAh | *his/its two* | كتاب *ktāb* | Nap | *writing* |
| ات *-āt* /At | NSuff-At | [fem-pl] | | | |

Table 3.2: Buckwalter lexicon entries sample [Buc04].

| Prefix-Stem | | Prefix-Suffix | | Stem-Suffix | |
|---|---|---|---|---|---|
| NPref-Al | N | NPref-Al | Suff-0 | PV | PVSuff-a |
| NPref-Al | N-ap | NPref-Al | NSuff-u | PV | PVSuff-ah |
| NPref-Al | N-ap_L | NPref-Al | NSuff-a | PV | PVSuff-A |
| NPref-Al | N/At | NPref-Al | NSuff-i | PV | PVSuff-Ah |
| NPref-Al | N/At_L | NPref-Al | NSuff-An | PV | PVSuff-at |
| NPref-Al | N/ap | NPref-Al | NSuff-ayn | PV | PVSuff-ath |

Table 3.3: Buckwalter compatibility table sample [Buc04].

#### 3.5.2.2 Xerox Arabic morphological analyzer

A morphological analyzer for Modern Standard Arabic was produced in 1996 at the Xerox Research Centre Europe [Bee01]. Xerox Morphology is based on solid and innovative finite-state technology [DF03].

This analyzer performs morphological analysis of orthographical words, whether fully diacriticised, partially diacriticised, or undiacriticised. Analyses show the root, pattern and all other affixes, together with part of speech tags which include information like person, number, gender, etc. The system was based on dictionaries from an earlier project at ALPNET [Bee90], and was extensively redesigned using Xerox finite state technology, resulting in an Arabic Finite State lexical Transducer.

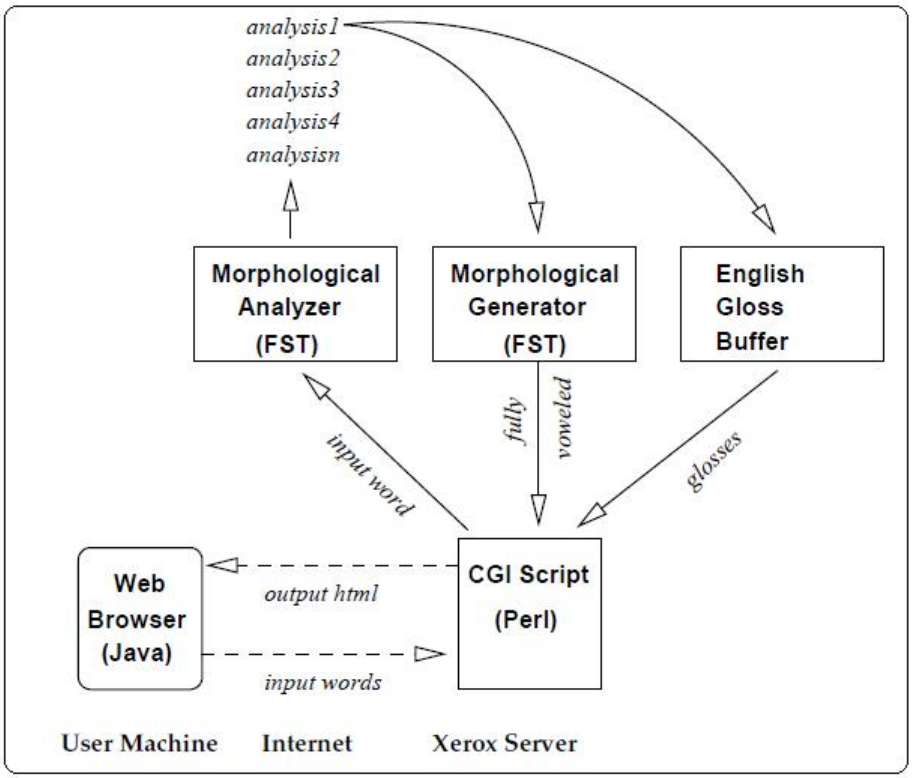



Figure 3.2: Xerox Arabic system [Bee98]

#### 3.5.2.3 ALMORGEANA

ALMORGEANA is a morphological analysis and generation system that used the Buckwalter Arabic Morphological Analyzer database. However, in the analysis stage, it generates an output in the lexeme and feature format rather than stem and affix format. Therefore, lexeme and feature keys are added to the lexicon which are used in analysis and generation [Hab10].

- **Analysis:** like the BAMA engine, words in ALMORGEANA are segmented into the format prefix-stem-suffix. The difference is in the step that uses lexeme

and feature keys that are associated with stem, prefix and suffix string sequence to generate the lexeme and feature format. Let us compare between the analysis of BAMA and ALMORGEANA on the same phrase لّكتب *llktb*. we will have the following analysis:

- **ALMORGEANA:** lilkutubi=[kitAb_1 POS:N1+Al+PL+GEN]=books
- **BAMA:** li/PREP+Al/DET+kutub/NOUN+i/CASE_DEF_GEN

the word lilkutubi is fully diacriticized which consists of the preposition لِ *li* 'for', the definite article ال *āl* 'the', the main lexemeكِتَاب *kitāb* 'book', the features PL for plural and GEN for genitive. It is worth noting that the feature PL is not present in BAMA because it is embedded in the word itself كُتُب *kutub* 'books'. So the feature PL is part of the extension done in ALMORGEANA in processing the BAMA database.

- **Generation:** The generation is the opposite process. The analyzer takes as an input a lexeme and feature set, and produces as an output a fully diacriticized word.

#### 3.5.2.4 MAGEAD

MAGEAD is a morphological and analyzer and generator for Arabic and its dialects [HR06]. It takes a lexeme and a set of linguistic features and produces surface word form through a sequence of transformations. In a generation perspective, the Arabic word is produced by:

- First creating a word stem from templatic morphemes.
- Then affixational morphemes are added to this stem.
- Applying the morphological and phonological rules. The process of combining the affixes with the stem involves a set of morphological and phonological rules. For example: applying the pattern rule iV1tV2V3 to the root زهر *zhr* will first give: ازتهر *āzthr*. However for pronunciation reasons, this is not allowed in Arabic. Therefore applying the rule will change ت *t* into د *d*, so the resulting word will be ازدهر *āzdhr* 'Flourish'.

#### 3.5.2.5 MADA+TOKAN

MADA (stands for Morphological Analysis and Disambiguation for Arabic) is a toolkit that takes as an input a raw Arabic text and recovers as much lexical and morphological information as possible. It produces in one operation POS tags, lexemes, diacritisations and full morphological analyses [HR09b]. A morphological analyser such as the ones we described (ALMORGEANA or MAGEAD), produces the different readings of a word regardless the context. However, a morphological disambiguation system such as MADA aims to produce the correct reading of a word in a specific context. Once a morphological analysis is decided, MADA can decide its full POS tag, lemma and diacritisation. ALMORGEANA is used internally to produce a list of potential analyses for each word of the sentence regardless their context. In order to choose the right analysis for each word, MADA makes use of up to 19 features to rank the list of analyses. Then, MADA uses a Support Vector Machine (SVM) classifier to create a prediction for the value of 14 features for each word in its context. The remaining features capture information such as spelling variations and n-gram statistics. Those analyses that more closely agree with the weighted set of feature predictions receive higher ranking scores. The correct analysis for each word in its context is the one with the highest score.

**TOKAN:** It is a general tokenizer for Arabic that takes MADA's output and tokenise it into a large set of possibilities. For example, MADA can decide whether a word is attached to a conjunction clitic. However, only until it comes to TOKAN, the tokenisation of the clitic is done. So TOKAN takes as an input a MADA disambiguated file and a tokenisation scheme that specifies the tokenisation target. If we are given the following specification:

- "w+ f+ b+ k+ l+ s+ Al+ REST + / + POS +P: +O: -DIAC"

و *w* and ف *f* are clitic conjunctions. ب *b*, ك *k*, ل *l* are prepositions. س *s* is a verbal particle. ال *āl* is the definite article.

So given the above specification and the phrase وسيعلمهم *wsyʕlmhm* 'he will teach them', it would be tokenised like: "wa+sa+yu'allim/V +hm"

### 3.5.3 MXL tagger

This is a tagger for undiacriticised MSA but was first trained on the fully diacritised Quran. This involved applying a rule based tagger on the Quran. A set of the output

is then manually corrected. The results of this correction is then used as a training data for an undiacriticised text. The rule based tagger involves matching the beginning and the end of a word. Since Arabic is highly inflectional and derivational language, matching the start and the end of the words means that the prefixes, suffixes and clitics will help in deciding the tag. A Transformation Based Learning (TBL) approach was used to improve the original set of rules [RS09]. We have converted the application of this tagger in prolog so that it can be integrated with the parser.

# Chapter 4

# Background: Parsing

## 4.1 Overview

This chapter consists of two parts:

1. First we describe Natural Language Parsing in general including the different models and algorithms. We then investigate and compare between different algorithms such as (top-down vs bottom up), (Breadth-first search vs depth-first search), and (Early vs CYK) algorithms. By describing the properties of these algorithms, we can show the appropriate one for our task. Then we describe statistical parsing and compare between two famous models Charniak and Collins.

2. The second part introduces Modern Standard Arabic parsing.

## 4.2 Parsing algorithms

Parsing is an important step in Natural Language Processing. Parsing is the process of recognizing an input string and producing some sort of structure for it [JM00]. There are different types of parsing, and this is solely based on the type of structure that can be produced: morphological, syntactic, semantic and pragmatic parsing, in the form of a string, a tree, or a network. In other words, the task of the parser is to take a sentence as an input and return a syntactic representation that corresponds to the likely semantic representation of that sentence. For example: given a sentence like : ”The boy is sleeping” would return a parse tree as shown in Figure 4.1

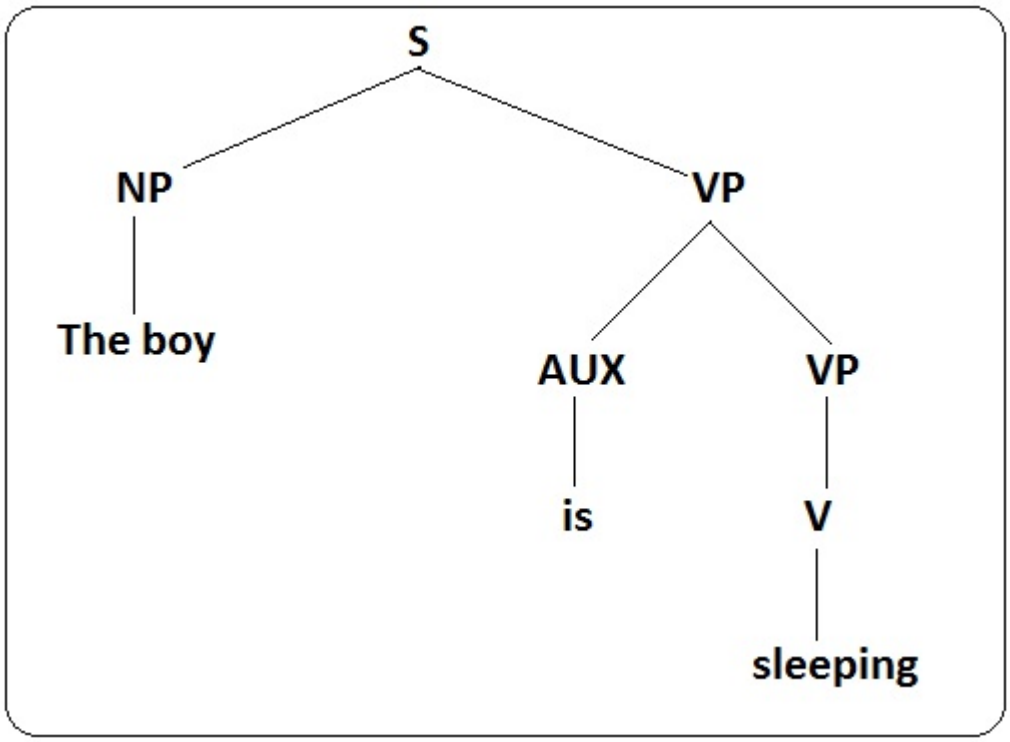



Figure 4.1: An example of a parse tree [JM00]

### 4.2.1 Parsing as Search

The parsing task can be viewed as a search problem. For instance, in syntactic parsing, the parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence [JM00]. For example given an English sentence like ”Book that flight”, and the small grammar and lexicon below, then the correct parse tree that would be assigned to this sentence is shown in Figure 4.2.

This is regarded as a search because we are trying to find all trees whose root is the start symbol S, and cover exactly the words in the input.

### 4.2.2 Chart parsing

General search methods are not best for syntactic parsing because the same syntactic constituent may be rederived more than one time as a part of different larger constituents, for instance, an NP could be part of different VPs or PPs. To solve this

| | |
|---|---|
| S → NP VP | |
| S → Aux NP VP | |
| S → VP | Det → that \| this \| a |
| NP → Det Nom | Noun → book \| flight \| meal \| money |
| NP → Proper-Noun | Verb → book \| include \| prefer |
| VP → Verb | Aux → does |
| VP → Verb NP | Prep → from \| to \| on |
| Nom → Noun | Proper-Noun → Houston |
| Nom → Noun Nom | |
| Nom → Nom PP | |

Table 4.1: English grammar and lexicon [JM00]

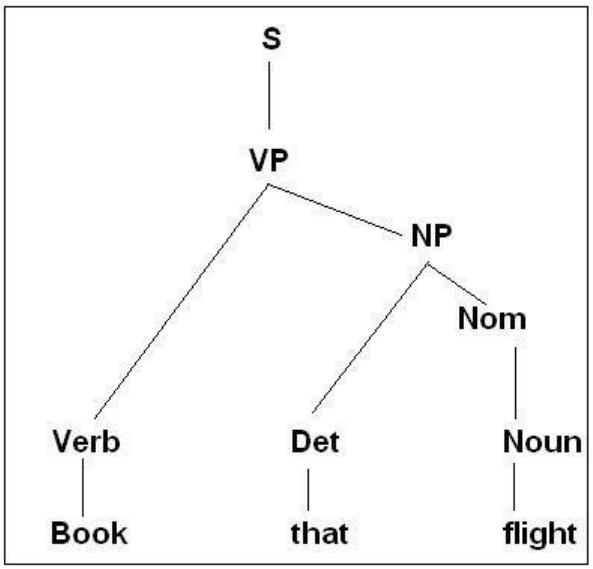



Figure 4.2: An example of parsing

problem, chart parsing uses a chart to keep track of partial derivations.

There are two kinds of search strategies for tree search: breadth-first search and depth-first search.

#### 4.2.2.1 Breadth-first vs depth-first search

Breadth-first search: an uninformed search strategy whereby the search space is explored by visiting all neighboring (sister) nodes first, before going deeper into the tree. i.e. it explores all branches at each level before going to the next level, whereas depth-first search is an uninformed search strategy whereby the search space is explored by going deeper and deeper (down a branch of the tree structure) until backtracking is required.

#### 4.2.2.2 Top-down parsing

Top-down parsing is a strategy that searches for a parse tree by starting with the root node (for example S in our previous example), and progress towards the leaves. For example, in our previous example, the parser assumes that the input can be derived by the start symbol S. By looking at all grammar rules with S on the left-hand side, there are three rules that expand S. Therefore the next level has three partial trees. We then expand the new items of the tree (NP and VP) just as we originally expanded S. At each level of the search space we use the right-hand sides of the rules to provide new sets of expectations for the parser, which are then used to recursively generate the rest of the trees. When the tree is fully built, leaves which fail to match all words in the input will be rejected.
In other words, parsing the sentence ”The boy sleeps”, the parser says:

- I am looking for an S.
- To get an S, I need an NP and a VP.
- To get an NP, I need a DET and an N.
- To get a DET, I can use ”The”.
- To get an N, I can use ”boy”.
- This completes the NP.
- To get a VP, I need a V.

- To get a V, I can use ”sleeps”.
- This completes the VP.
- This completes the S.

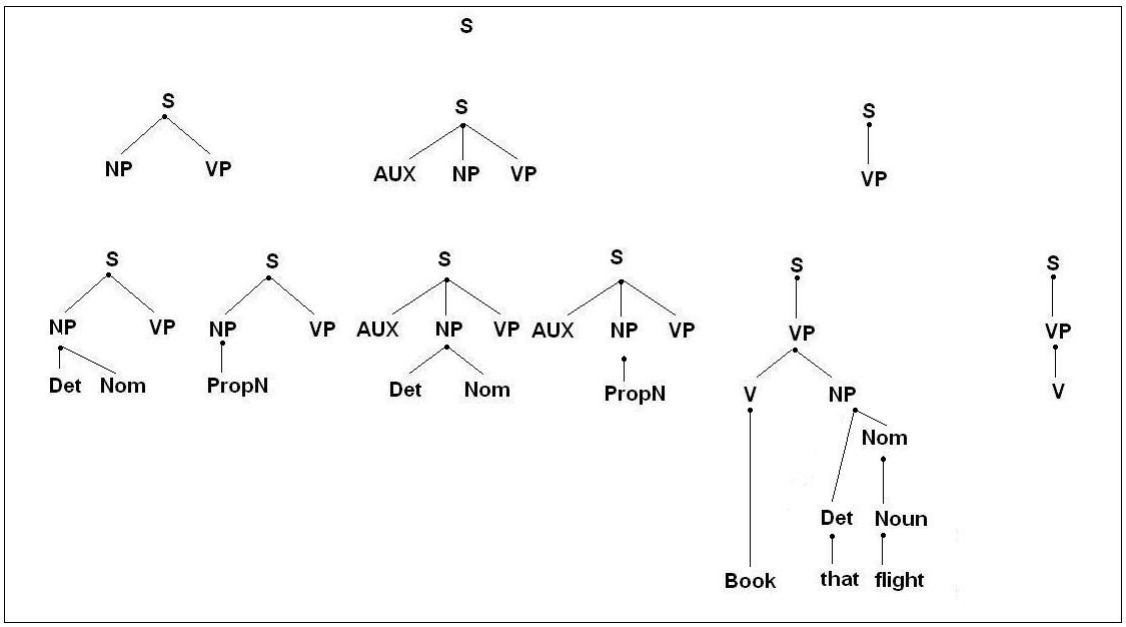



Figure 4.3: An example of top down parsing [JM00]

#### 4.2.2.3 Bottom-up parsing

Unlike the top-down parsing, a bottom-up parser starts with the words of the input, and tries to progress towards the goal (the full parse tree). The parse is successful only if the parser succeeds in building a tree whose root is the start symbol S and covers all of the input. The basic operation of bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules, then replace these sequence of symbols by its left-hand side symbol. Figure 4.4 shows the bottom-up search space beginning with the sentence ”Book that flight”. The parser starts by looking up each word in the lexicon (book, that, flight). Because book is an ambiguous word (it can be a noun or a verb), the parser will consider two sets of trees. We then, apply the rule NOM $\rightarrow$ Noun to both of the nouns (book and flight) to produce the third level. We do the same in the next levels, by looking at the rules and see if the right-hand side of some rule might fit

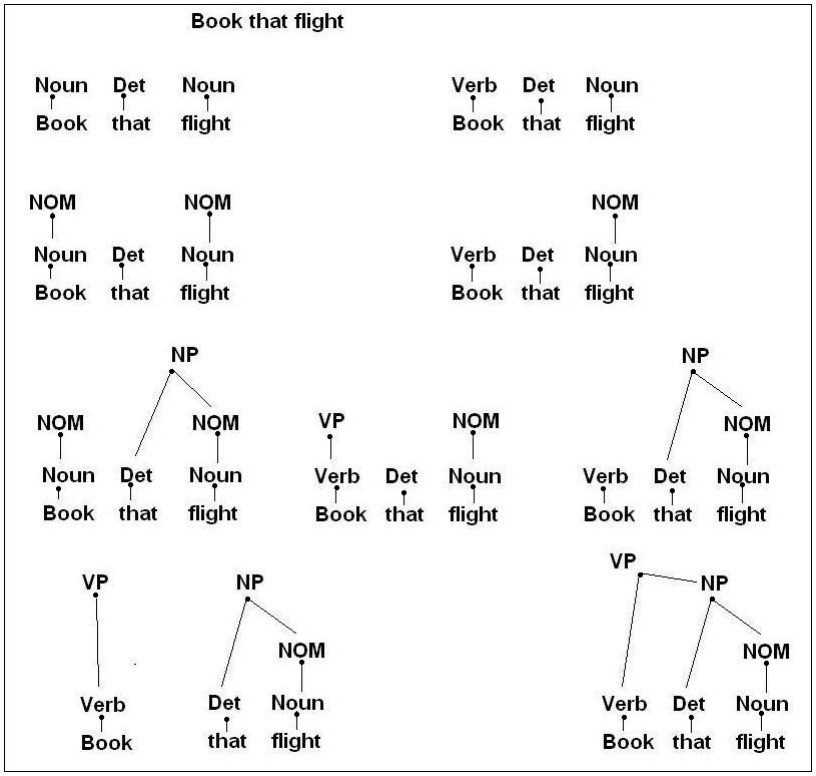



Figure 4.4: An example of bottom up parsing

#### 4.2.2.4 Top-down parsing vs Bottom-up parsing

The main difference between bottom-up and top-down parsers is the way the grammar rules are used. Each of these two search strategies has its own advantages and disadvantages. The top-down strategy does not waste time in exploring parses that cannot result in an S, whereas in the bottom-up strategy, trees with no hope of leading to an S are generated and later on abandoned. This is considered as wasted effort. For example, considering "book " as a noun at the beginning of a sentence, despite the fact that there is no rule in the grammar that will lead to an S based on this grammar. However, if the top-down approach does not waste time in exploring trees that do not lead to an S, it does spend a considerable effort on trees that will not match the input. This inefficiency in top-down strategy is due to the fact that the parser generates trees even before examining the input. On the other hand bottom-up strategy never produces trees that are not at least existent in the actual input.

There are three problems that face top-down and bottom-up parsers: left recursion, ambiguity, and inefficient reparsing of subtrees [JM00]. An example of left recursion is (NP $\rightarrow$ NP VP), which will lead to indefinite expansion of trees for top-down parsing. Ambiguity is when words may have more than one part of speech (lexical ambiguity), when the grammar assigns more than one possible parse to a sentence (structural ambiguity), when a particular constituent can be attached in more than one place in a parse tree (Attachment ambiguity), or when different constituents can be formed from a conjunction (Coordination ambiguity). Inefficient reparsing is when the parser builds valid trees, and then discards them during backtracking, only to find that it has to rebuild them again, or attempts to build invalid ones .
Lexical ambiguity is worse for bottom up parsing whereas structural ambiguity is worse for top-down parsing

In order to solve these problems, dynamic programming algorithms are used.
Dynamic programming is a method used to discover solution to subproblems along the way to the solution of the main problem. It is generally good for problems with overlapping subproblems. The most famous dynamic programming algorithms for parsers are: the Early algorithm and the CYK algorithm.

#### 4.2.2.5 Early algorithm

Early's algorithm [Ear70] is considered as a breadth-first top-down parser with bottom-up recognition. This algorithm reduces an apparently exponential-time problem to a

polynomial-time one by eliminating the repetitive solution of subproblems. The Early algorithm produces a table called ”chart” that has N+1 entries, each of which contains a list of states representing the partial parse trees generated so far. By the end of the sentence, each possible subtree is represented only once and can therefore be shared by all the parses that need it.

```
repeat until input is exhausted
  a = /* current input symbol */
  k = /* current state index */

  repeat until no more states can be added
    foreach state (X ::= A•YB, j) in state[k]        // prediction
      foreach rZule (Y ::= C)
        state[k].add( state (X ::= •C, k) )

    foreach state (X ::= A•aB, j) in state[k]        // scanning
      state[k+1].add( state (X ::= Aa•B, j) )

    foreach state (X ::= A•, j) in state[k]          // completion
      foreach state (Y ::= A•B, i) in state[j]
        state[k].add( state (Y ::= A•XB, i) )
```

Figure 4.5: Early algorithm [JM00]

#### 4.2.2.6 CYK algorithm

The Cocke-Younger-Kasami (CYK) is a bottom-up dynamic programming parsing algorithm. The standard version of CYK recognizes languages defined by context free grammars written in Chomsky Normal Form CNF. Because any context free grammar can be converted to CNF, CYK can be used to recognize any context free language. The standard algorithm can also be extended to accept forms that are not CNF, but at the cost of making the algorithm more difficult to understand.

However, both Early and CYK algorithms are not suitable to handle Modern Standard Arabic, because of its relatively free word order. What is suitable for languages with free word order including MSA is a variation on chart parsing implemented by Allan Ramsay [Ram99]. This will be discussed later in the thesis.

```
int N = /* number of input tokens */
int R = /* number of rules in CNF grammar */

bool array[N][N][R];

foreach token T at index I in the input
  foreach rule R -> T
    array[I][1][R] = true

foreach I = 2..N
  foreach J = 1..N-I+1
    foreach K = 1..I-1
      foreach rule R -> S T
        if P[J][K][S] and P[J+K][I-K][T]
          P[J][I][R] = true
```

Figure 4.6: CYK algorithm [JM00]

## 4.3 Statistical parsing

An important aspect of this thesis is to collect statistics about edges and use these to guide a rule based parser. That is why exploring the approach of statistical parsing is introduced in this section.

Syntactic ambiguity is a serious problem and also it is very hard to manually define a grammar whose rules determine a single parse for any given sentence. That is why statistical models were introduced as they provide methods for selecting between the alternative parses. In the early days, this involved the use of manually parsed training material to obtain scores for local tree configurations, which were combined to obtain a score for a complete parse-tree. The release of the Penn Treebank led to a large interest in Statistical parsing, and started a parsing competition among the NLP community. Since then, There has been a great deal of interest in exploiting corpus resources for different Natural Language Processing (NLP) tasks. Magerman [Mag95] was one of the earliest to release parsing accuracies on the Penn Treebank. His parser was called SPATTER which is based on decision-tree learning techniques. A decision tree is a decision-making device which assigns a probability to each of the possible choices based on the context of the decision [Mag95]. These probabilities are estimated using statistical decision tree models. The probability of a complete parse tree of a sentence is the product of each decision conditioned on all previous decisions. SPATTER achieved an accuracy of 84.3 percent labeled precision and 84.0 percent labeled recall [CFL10]. In 1996 this score was improved by Collins to 85.7 percent and 85.3 percent labelled precision and recall. A year later, Collins improved his results

further by introducing a new motivated approach based on a generative model [Col97].

## 4.4 Treebanks

A Treebank is a text corpus in which each sentence has been annotated with syntactic structure. This structure is usually represented as a tree structure, hence the name Treebank.
Treebanks are very important resources for building statistical parsers and evaluating them. Natural language processing, speech recognition, and other linguistic tools are in great need of large annotated corpora. The latter can be valuable for providing statistical data to probabilistic parsers and other natural language processing tools. Treebanks may also be beneficial for information retrieval and other traditional linguistic research. They can also serve in computational linguistics for training or testing parsers. Treebanks are often created on top of a corpus that has already been annotated with part of speech tags. But treebanks are sometimes enhanced with semantic or other linguistic information. They can either be created completely manually, where linguists annotate each sentence with a syntactic structure, or semi automatically where a parser assigns some syntactic structure which linguists can check and where necessary correct.

### 4.4.1 The Penn Treebank

One of the largest and widely used English-language treebanks is ”the Penn Treebank”, which was developed by Mitchell Marcus and his team at the University of Pennsylvania. It has played a great role in creating or improving many English natural language processing resources. Since it is well-documented, it has served as a solid methodology for researchers attempting to produce similar treebanks in other languages. The Penn Treebank is a large annotated corpus consisting of over 4.5 million English words annotated with both part-of-speech and syntactic tree information [MMS93]. It provides a large corpus of syntactically annotated examples mostly from the wall street journal.

## 4.5 Statistical parsing models

This section reviews the concept of probability theory which is the basis of the models used in statistical parsing. Some statistical models will be discussed. Statistical language models are used to capture some important properties of a natural language by collecting statistics from corpora of that language. For this purpose, various language models have been used such as: word N-grams.

### 4.5.1 N-grams model

As some of the parsers discussed later will use N-gram models to compute the probabilities, a brief introduction will be introduced in this section. The N-gram model is one of the most important tools in *NLP*. By assigning a conditional probability to possible neighboring words, N-grams can be used to assign a joint probability to an entire sentence [JM00].
One way of computing the probability is from relative frequency counts. $P(w|h)$ is the probability of $w$ given some history $h$. I.e. answering the question: "Out of the times we saw the history h, how many times was it followed by the word w". While estimating probabilities directly from counts works fine in many cases, It turns out that this is not enough to give good estimates. Therefore cleverer ways of estimating probabilities were introduced.

#### 4.5.1.1 Chain rule of probability

One way is to use the **chain rule of probability** to decompose the probability of an entire sequence of words [JM00].

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)...P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned} \tag{4.1}$$

The chain rule shows the relationship between computing the joint probability of a sequence and computing the conditional probability of a word $w$ given previous words. However this is not enough, because it is hard to estimate by counting the number of times every word occurs following every long string. Natural languages are creative and any particular context might have never occurred before.

Therefore the notion of **N-grams** is introduced here. Instead of estimating a probability given the entire history, we will approximate the history by just the last few words. The **Bigram** for instance only computes the conditional probability of the preceding word. Formally, using a bigram model the probability is estimated as follows:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{4.2}$$

So a bigram looks one word into the past, and trigram looks two words into the past, and thus, N-gram looks $N-1$ words into the past which gives the equation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{4.3}$$

#### 4.5.1.2 Maximum Likelihood Estimation

One of the simplest way to estimate probabilities is by dividing the counts for one item by the total counts, then normalising it so that it lies between 0 and 1. For example, if you have a corpus of 1 million words, and the word *chinese* appears 400 times. The probability estimate here is simply 0.0004. Estimating a bigram probability of a word $w$ given a previous word $h$:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\Sigma_w C(w_{n-1}w)} \tag{4.4}$$

The problem with MLE is that it predicts that the probability of a word that we have not seen is exactly zero. Also MLE is not going to give us a very useful estimation because it is simply using relative frequency counts. The problem gets worse if we are trying to use counts of one corpus to estimate observations of a different corpus. From the previous example the probability of any random word selected out of this corpus being the word chinese is $\frac{400}{1000000} = 0.0004$ which might not be accurate enough in an other corpus.

The trouble with MLE can also happen when we are dealing with a language that has lots of rare words. If we consider a language with 10000 rare words, and in a corpus we only encounter 10 rare words. This means that we have not seen 9990 rare words. That is why, MLE will be modified slightly to get better estimates.

#### 4.5.1.3 Smoothing

Sparse data problems in statistical parsing arise from the fact that the training data cannot cover all of the cases the parser will face [JM00]. A parser might encounter a word used as the head of an *NP*, but previously was only seen as an adjective. Any corpus is limited and it is possible that a perfectly accepted English sequence of words is going to be missing from it. This will result in many cases of Zero probability N-grams. Just because an event has never been observed in training data does not mean it cannot occur in test data. Therefore smoothing these probabilities is used to solve such problems. Smoothing is the process of flattening a probability distribution implied by a language model so that all reasonable word sequences can occur with some probability. There are different algorithms for smoothing:

- **Laplace smoothing:** One way to do smoothing is to add one to all the count before normalising.

  $P_{MLE}(w_i) = \frac{C(w_i)}{\Sigma_j C(w_j)} = \frac{C(w_i)}{N}$
  $P_{Laplace}(w_i) = \frac{C(w_i)+1}{\Sigma_j (C(w_j)+1)} = \frac{C(w_i)+1}{N+V}$

- **Good Turing Discounting:** The idea here is to use the count of things we have seen once to help estimating the count of things we have never seen. These estimators are based on theory which is correct for a large corpus of a large language. The Good Turing algorithm is based on computing $N_c$, the number of N-grams that occur $c$ times which is refered to as frequency of frequency. Formally: $N_c = \Sigma_{x:count(x)=c}$ 1. Given that The MLE count for $N_c$ is $c$. The Good-Turing estimate replaces this with a smoothed count $c^*$, as a function of $N_{c+1}$ as shown:

  $c^* = (c+1)\frac{N_{c+1}}{N_c}$

  Estimating probabilities of things that had zero counts $N_0$ as:

  $P^*_{GT}(things\ with\ zero\ counts\ in\ training) = \frac{N_1}{N}$
  where $N_1$ is things with 1 count.

- **Interpolation:** If we are trying to compute $P(w_n|w_{n-1}\,w_{n-2})$, but this particular trigram $w_{n-2}\ w_{n-1}\ w_n$ does not exist, we can estimate its probability by using

the bigram probability $P(w_n|w_{n-1})$. Similarly, if we do not have counts to estimate this bigram $P(w_n|w_{n-1})$ , we can look to the unigram $P(w_n)$. Interpolation combines the different order N-grams by linearly interpolating all the models. Formally estimating $P(w_n|w_{n-1}\,w_{n-2})$ by adding together the unigram, bigram, and trigram probabilities, each weighted by $\lambda$:

$$\begin{aligned}\hat{P}(w_n|w_{n-1}\ w_{n-2}) \approx \lambda_1 P(w_n|w_{n-1}\ w_{n-2}) + \\ \lambda_2 P(w_n|w_{n-1}) + \\ \lambda_3 P(w_n)\end{aligned} \tag{4.5}$$

such that: $\lambda_1 + \lambda_2 + \lambda_3 = 1$

## 4.5.2 Collins parsers

### 4.5.2.1 Collins first statistical parser

After the invention of the Penn Treebank, one of the most influential work was introduced by Collins [Col96]. This was a statistical parser based on probabilities of dependencies between head-words in the parse tree. This parser takes a tagged sentence as input and produces a phrase-structure tree as output. The statistical model assigns a probability to every parse tree for this sentence. In a formal way, given a sentence $S$ and a tree $T$, the statistical model estimates the conditional probability $P(T|S)$. The most likely parse is the one with the highest probability:

$$T_{best} = \arg\max_T P(T|S) \tag{4.6}$$

The idea of this statistical model is that any parse tree such as the one in Figure 4.7 can be represented as a set of baseNPs $B$ and a set of dependencies $D$. In other words: $T = (B, D)$
So the formula $P(T|S)$ will be represented as follows:

$$P(T|S) = P(B,D|S) = P(B|S) \times P(D|S,B) \tag{4.7}$$

Then given a sentence $S$ and baseNPs $B$ all punctuation will be removed and the baseNPs will be reduced to their head-word alone such as in Figure 4.8. Each constituent of the parse tree will get its head-child from its children using some simple rules. For example the rule:

$NP \rightarrow DET\ NN$

would identify $NN$ to be the head-child of $NP$.
And the rule:

$S \rightarrow NP\ VP$

would identify $VP$ as the head-child of $S$.
Each parent will receive its head-word from its head-child. For example: given the rule

$S \rightarrow NP\ VP$

$S$ would get its head-word *announced* from its head-child $VP$.
Head-modifier relationships can be extracted as shown in Figure 4.8 , where each constituent contributes a set of dependency relationships.

#### 4.5.2.2 The Generative Models

These are three generative, lexicalised Probabilistic Context-Free Grammar(PCFG) parsing models [Col97]. Model 1 is a generative version of the one described in [Col96]. PCFGs were extended to lexicalised grammars that associate a head word (a word and a POS tag) with each non terminal in a parse tree. In Model 2, the parser was extended to make the complement/adjunct distinction by adding probabilities over subcategorisation frames for head-words. In Model 3, wh-movement was introduced using a method similar to the slash feature in the Generalized Phrase Structure Grammar [GKPS85].

#### 4.5.2.3 Collins probability model

The basis in statistical parsing is associating probabilities with grammatical rules. The probability of any given parse tree is the product of the probabilities of all the rules applied in creating this parse tree. Formally, for each rule $LHS \rightarrow RHS$:

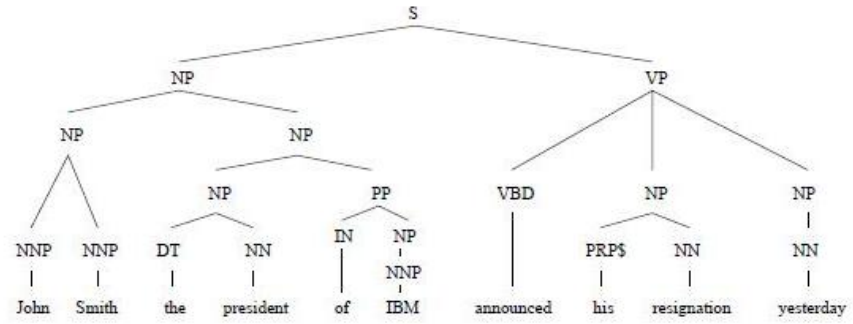
$$P(RHS|LHS) = \frac{count(LHS \rightarrow RHS)}{count(LHS)} \tag{4.8}$$

The product of the probability of individual rules is used to calculate the joint probability of a tree $T$ and a given sentence $S$:

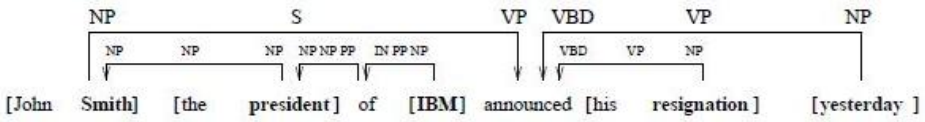
$$P(T,S) = \prod_{i=1}^{n} P(LHS_i|RHS_i) \tag{4.9}$$

(1) John/NNP Smith/NNP, the/DT president/NN of/IN IBM/NNP, announced/VBD his/PRP$ resignation/NN yesterday/NN .

(2)




(3)




(4)

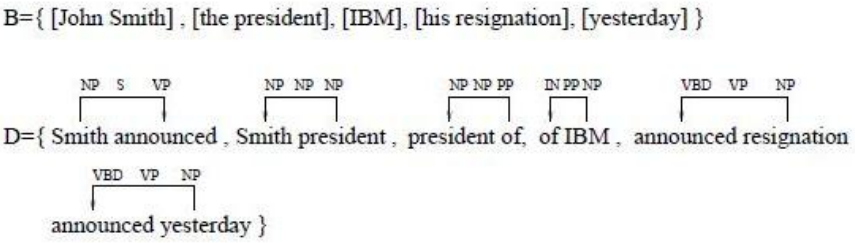



Figure 4.7: A representation used in Collins model. (1) is a tagged sentence, (2) is a parse tree, (3) is a dependency representation, (4): B is the set of baseNPs, D is the set of dependencies [Col96]

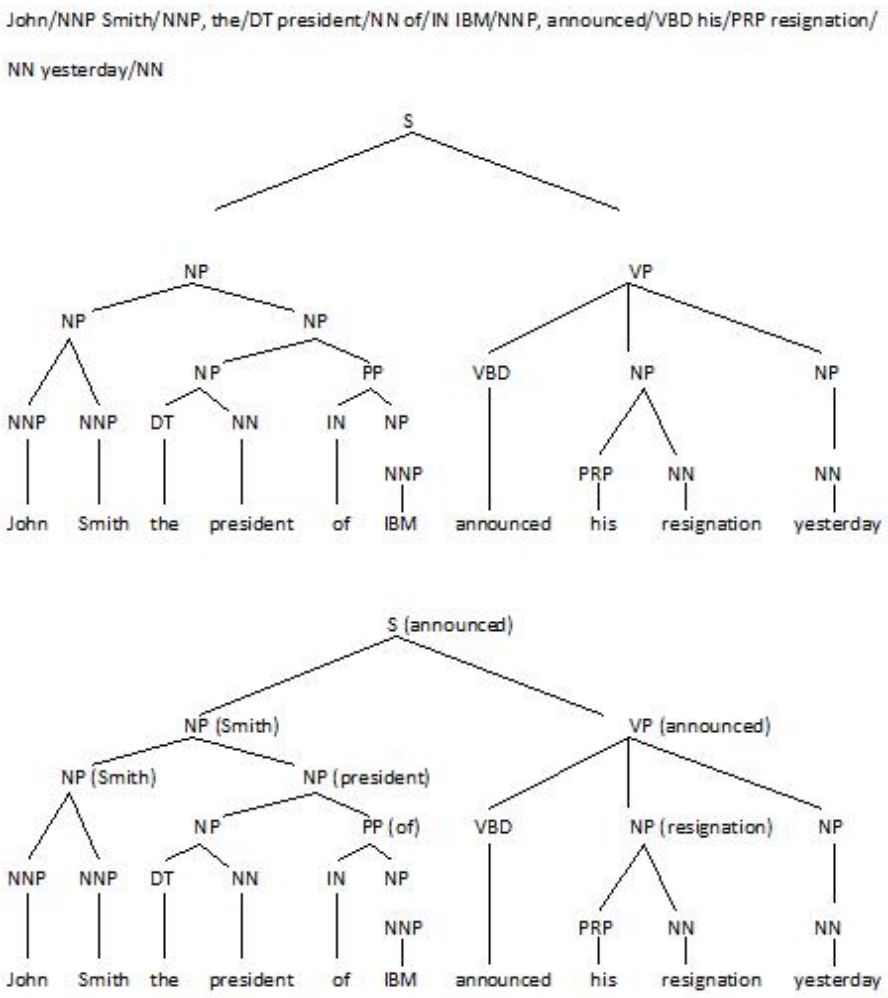



Figure 4.8: A parse tree with Head-Child dependencies [Col96]

The best parse tree is the one with the highest probability:

$$T_{best} = \arg\max_T P(T|S) = \arg\max_T \frac{P(T,S)}{P(S)} = \arg\max_T P(T,S) \tag{4.10}$$

Given the rule $S \rightarrow NP\ NP\ VP$ and using equation 4.8 we can estimate the probability:

$$P(NP \quad NP \quad VP|S) = \frac{count(S \rightarrow NP \quad NP \quad VP)}{count(S)} \tag{4.11}$$

When the grammar is lexicalised, i.e. associating each non-terminal with a head-word and a pos tag. Given the example mentioned in [Col03] "Last week IBM bought Lotus" which its parse tree is in Figure 4.9.

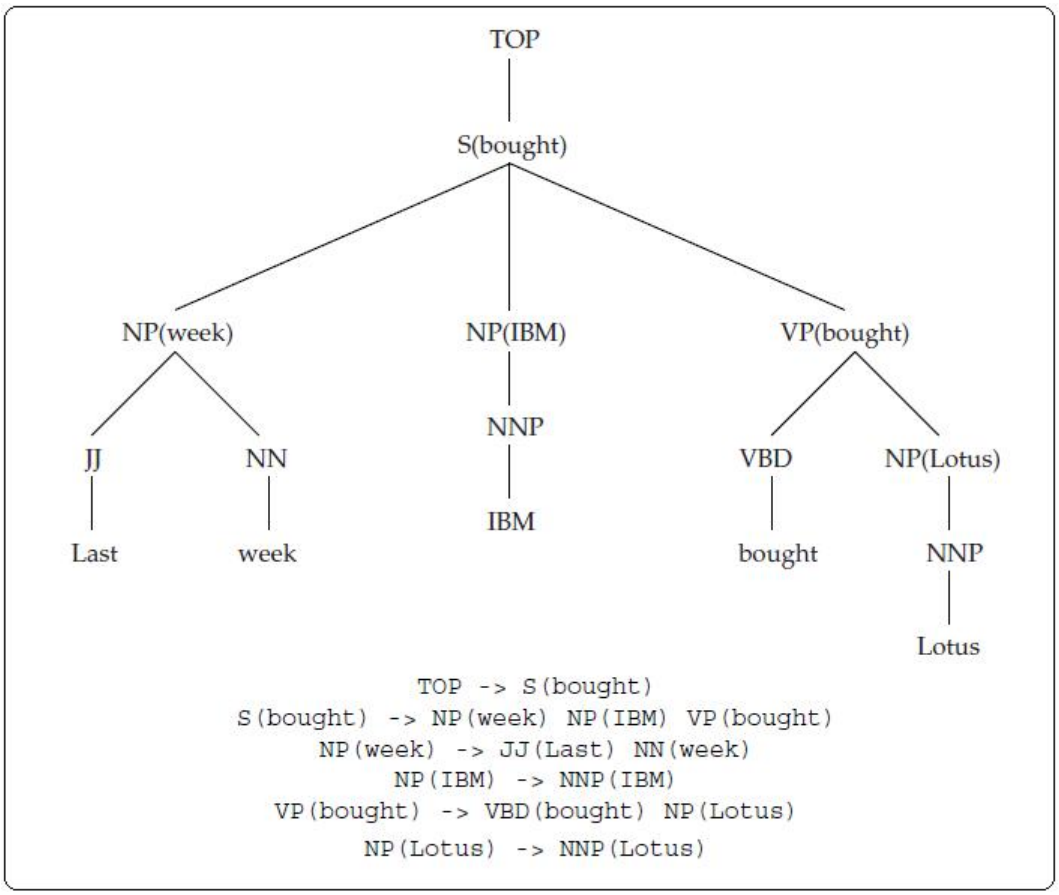



Figure 4.9: A parse tree example from [Col03]

The rule $S \rightarrow NP\ NP\ VP$ would become:

$$S(bought,VBD) \rightarrow NP(week,NN) \quad NP(IBM,NNP) \quad VP(bought,VBD) \tag{4.12}$$

By applying equation 4.11 the probability is estimated as follows:

$$P(T|S) = \frac{count(S(bought,VBD) \rightarrow NP(week,NN)\,NP(IBM,NNP)\,VP(bought,VBD))}{count(S(bought,VBD))} \tag{4.13}$$

The use of lexicalisation in this way will result in an inaccurate estimation of the probability. Because for PCFG estimating the probability is simply counting the number of times LHS is seen expanded as the RHS divided by the total number of times the LHS is seen in the training corpus. However, lexicalisation will give a huge a number of non-terminals resulting in a severe sparse data problem. A lexicalised PCFG may result in two problems:

- The LHS of a rule associated with a head word may not appear in the training data.

- Even if the LHS appears in the data, it may never expand to the RHS.

An example of this problem:
The rule *S* headed by the verb *bought* which results in an *NP* headed by *week*, an *NP* headed by *IBM* and a *VP* headed by *bought* is not going to appear very often in the corpus. The addition of lexical heads will result in a huge number of rules of this type. Therefore the numerator of most rules of this kind is likely going to be 0.
Collins solution to this problem is to break up the generation of the rule into parts and calculate the probability of each part separately. The RHS is generated from the head outwards, with first-order Markov processes separately generating the modifiers to the left and right of the head [Col03]:

$$PR(h) \rightarrow L_n(l_n)...L_1(l_1)\,H(h)\,R_1(r_1)...R_m(r_m) \tag{4.14}$$

$H$ is the head child of the phrase which inherits the head-word $h$ from its parent $P$. $L_n...L_1$ are its left modifiers, and $R_1...R_m$ are its right modifiers. Taking the rule in 4.14, the generation of the RHS given the LHS can be decomposed into three steps:

1. Generate the head constituent label of the phrase, with probability:

$$P_H(H|PR,h) \tag{4.15}$$

2. Generate modifiers to the right of the head with probability:

$$\prod_{i=1}^{m+1} P_R(R_i(r_i)|PR,h,H) \tag{4.16}$$

where $R_{m+1}(r_{m+1})$ is defined as the stop

3. Generate modifiers to the left of the head with probability:

$$\prod_{i=1}^{n+1} P_L(L_i(li)|PR,h,H) \tag{4.17}$$

where $L_{n+1}(l_{n+1})$ is defined as the stop

Applying these equations to the rule:
$S(bought,VBD) \rightarrow NP(week,NN) \quad NP(IBM,NNP) \quad VP(bought,VBD)$, the probability will be estimated as:

$$\begin{aligned} &P_h(VP|S,bought) \times P_l(NP(IBM)|S,VP,bought) \times \\ &P_l(NP(week)|S,VP,bought) \times P_l(STOP|S,VP,bought) \times \\ &P_r(STOP|S,VP,bought) \end{aligned} \tag{4.18}$$

This will result in a useful and more effective probability. So Collins work here allows including lexical information while still estimating a useful probability.

### 4.5.3 Charniak's model

Charniak presents a statistical parser that induces its grammar from a hand-parsed corpus. Charniak claims that although the common wisdom has it that grammar obtained from treebanks do not perform well, his parser outperforms all other non-word-based statistical parsers. Charniak parser takes a sentence and returns the most probable parse (the parse with the highest probability). It is trained on the Penn treebank. It reads a Context Free Grammar as described in [CCA+96], then collect statistics used

to compute the empirically observed probability distributions. A new test sentence is parsed to obtain a set of parses using the context free chart parsing technique. There is no attempt of finding all parses. Only constituents that promise to contribute to the most probable parse are selected (the improbable parses are ignored). This parser was improved in 1999 to achieve 90% precision/recall for sentences of length 40 and less, and 89.5% for sentences of length 100 and less, when trained on the standard sections of the Wall street journal treebank. Charniak used a technique called "Maximum-Entropy-Inspired" model for conditioning and smoothing [Cha]. The advantage of this technique is that the features used for encoding the different conditional probability events do not have to be statistically independent of each other. In addition, the generation of a modifier is conditioned on its two previous sisters.

#### 4.5.3.1 The probability model

The probabilistic model presented by Charniak takes a sentence $s$ and returns the parse with the highest probability $p(\pi|s)$. Formally, the parser returns $P(s)$ where:

$$P(s) = \arg\max_{\pi} \frac{p(\pi, s)}{p(s)} = \arg\max_{\pi} p(\pi, s) \tag{4.19}$$

We consider the example mentioned in [Cha97]: "Corporate profits rose" which one of its parse trees is in Figure 4.10:

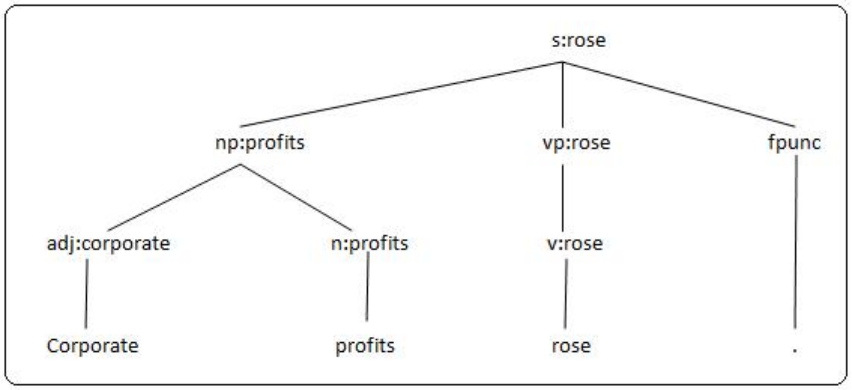



Figure 4.10: A parse tree

Since this is a PCFG, the tree in Figure 4.10 can be rewritten as a set of CFG rules specifying how each constituent is expanded. Each non-terminal is headed by a word which is the head. The head can be determined by some simple rules in a bottom up way, such as the head of an *np* is the main *noun*, the head of a *vp* is the main *verb*, and

the head of a sentence is the head of the sentence's *vp*. Estimating the probability of a constituent in a top down fashion will include:

1. Calculate the probability of the head
2. Calculate the probability of a constituent given the head
3. Recurse down in parse tree to estimate probabilities of sub-constituents.

Given the *np* "corporate profits"and assuming the head $s$, its type $t$, the head of its parent $h$, and the type of the parent $l$, we need to compute $p(s|h,t,l)$. For example: *corporate profits:np* will be $p(profits|rose,np,s)$.
Approximating $p(s|h,t,l)$ using the deleted interpolation formula will give:

$$\begin{aligned} p(s|h,t,l) = \lambda_1(e)\hat{p}(s|h,t,l)+ \\ \lambda_2(e)\hat{p}(s|c_h,t,l)+ \\ \lambda_3(e)\hat{p}(s|t,l)+ \\ \lambda_4(e)\hat{p}(s|t) \end{aligned} \tag{4.20}$$

### 4.5.4 Comparison of Charniak and Collins work

Both Collins and Charniak used the Penn Wall Street Journal Treebank to train their parsers. They both used the same training and testing data. They both collected statistics from the training corpus, using the notion of lexical heads. Both works restrict head information to two levels, i.e. collecting statistics about the head and its parent. In addition both parsers take the correct parse the one with the highest probability. However, there are some differences in terms of grammar, the statistics collected and smoothing. Charniak estimate $p(s,\pi)$, whereas Collins compute $p(\pi,s)$. Charniak has an explicit grammar which is extracted from a treebank. They are also different in the way smoothing was conducted.

## 4.6 Modern Standard Arabic (MSA) parsing

After we have introduced NLP parsing in general, in this section we illustrate parsing MSA and compare the few tools available for the language. This includes comparing the different MSA treebanks and parsers.

### 4.6.1 Arabic treebanks

The most well known Arabic treebanks are The Penn Arbic Treebank (PATB), The Prague Arabic Dependency Treebank (PADT) and the Columbia Treebank (CATiB). It is worth noting that both PATB and PADT employ very complex linguistic representations that require a lot of human efforts. This makes the representations of these two treebanks full of details, which is useful when producing POS tags, in tokenisation, syntactic structures, and other specific details.
CATiB, on the other hand was produced for the purpose of speeding up the annotation process through representation simplification.

In this section, we present a brief discussion of these treebanks and do some comparison between them. Let us take the same sentence and see its representation in each of the three treebanks. خمسون ألف سَائِح زَارُوا لبنَان و سوريَا في أيلول المَاضي *ẖmswn ʾlf sāʾiḥ zārwā lbnān w swryā fy ʾylwl ālmāḍy* 'Fifty thousand tourists visited Syria and Lebanon last September'

#### 4.6.1.1 The Penn Arabic Treebank (PATB)

PATB was a result of a team work of Maamouri, Ann Bies, and Hubert Jin. They have started with the assumption that the existing methodological principles of the previous Penn treebanks could be very beneficial in constructing the Arabic Treebank [MB04]. The Buckwalter analyzer is used for the morphological annotation and POS tagging of the Arabic newspaper text. Therefore, for every input string, the analyzer provides a fully diacriticised solution in Buckwalter transliteration. At this step, human intervention is necessary to disambiguate many orthographically identical forms. An example of tree representation in the PATB is given in Figure 4.11: The creation of the PATB was very important for Arabic research. All other treebanks that were produced after the PATB have used it or at least used some of the tools developed for it. In fact the PADT and CATiB both converted the PATB representation to their own representation.

#### 4.6.1.2 Prague Arabic Dependency Treebank (PADT)

PADT is a project of the Institute of Formal and Applied Linguistics at Charles University in Prague.It is a younger sibling to Prague Dependency Treebank for Czech [HSZ$^+$04]. It consists of morphologically and syntactically annotated newspaper texts

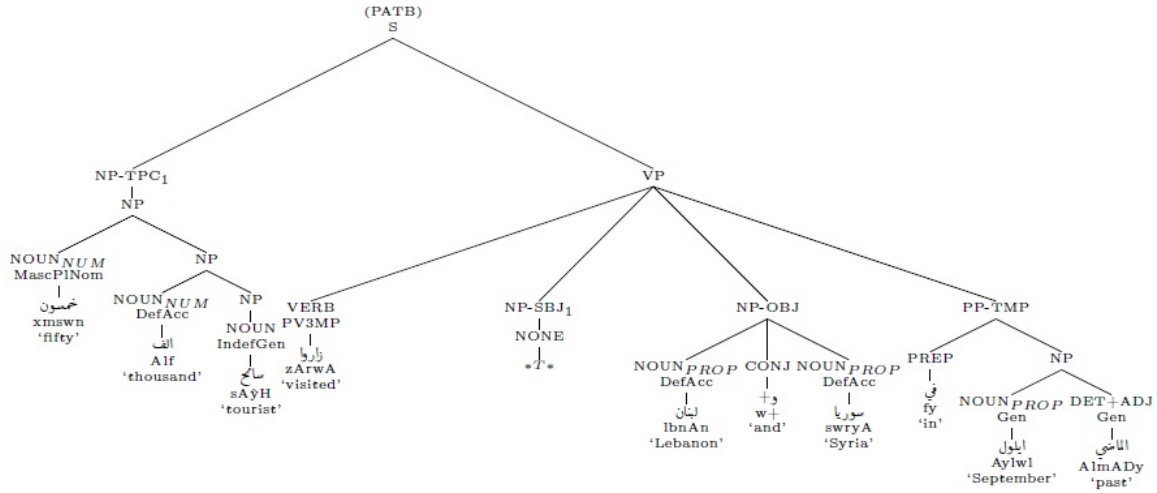



Figure 4.11: The PATB tree of **خمسون ألف سائح زاروا لبنان و سوريا في أيلول الماضي** *ḥmswn ʾlf sāʾiḥ zārwā lbnān w swryā fy ʾylwl ālmāḍy "Fifty thousand tourists visited Syria and Lebanon last September"* [Hab10]

of MSA, which originates from resources published by the Linguistic Data Consortium, University of Pennsylvania, Arabic Gigaword and plain data of Penn Arabic Treebank. The morphological and syntactic annotations in PADT is different from that of the PATB. An example of the same sentence in the PADT representation is given in Figure 4.12. The head of the sentence is the verb **زاروا** *zārwā* 'visited'. it has three children: A subject 'Sb', a coordinating conjunction 'coord' **و** *w* 'and', and an auxiliary prepositional phrase 'AuxP'. The subject itself is a number word modified by another number word, which is also modified by a noun **خمسون ألف سائح** *ḥmswn ʾlf sāʾiḥ* 'fifty thousand tourists'. The second child of the verb **و** *w* 'and' heads two proper nouns with the composite relation Obj_co, which means two nouns ara coordinated by their parent **و** *w* and they are both objects of their grandparent verb. The last child of the verb is **في** *fy* 'in' which heads a proper noun **ايلول** *āylwl* **سبتمبر** *śeptember* with the relation Adv (Adverbial) which in turn heads an adjective **الماضي** *ālmāḍy* 'past' in attribute relation Atr.

### 4.6.2 The Columbia Arabic Treebank (CATiB)

Another work that resulted in a database of annotated sentences is the Columbia Arabic Treebank. CATiB is different from the previous treebanks in putting an emphasis

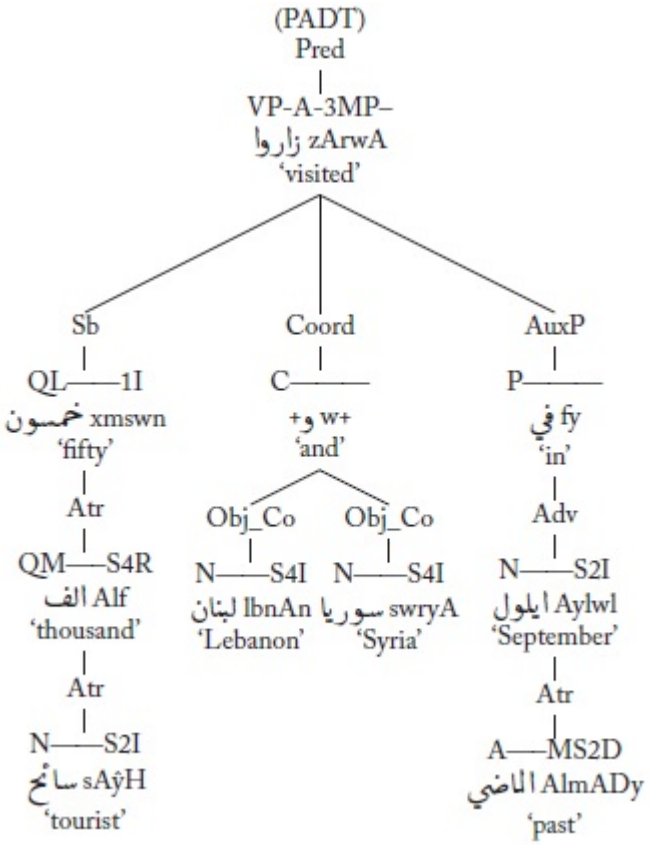



Figure 4.12: The PADT tree **خمسون ألف سَائِح زَارُوا لبنَان و سوريَا في أيلول المَاضي**
*hmswn ʾlf sāʾiḥ zārwā lbnān w swryā fy ʾylwl ālmāḍy "Fifty thousand tourists visited Syria and Lebanon last September"* [Hab10]

on annotation speed with some constraints on linguistic richness [HR09a]. CATiB approach was to avoid annotation of redundant linguistic information, for instance: nominal case and state in Arabic can be determined automatically from the syntax and morphological analysis of the words. Therefore, it is not necessary to be annotated by humans. Also CATiB uses dependency structure representation and relational labels induced from the traditional grammar. The idea behind this, is to make it easier to train annotators who do not need to have degree in linguistics.

Although CATiB uses the same basic tokenisation scheme, the POS tagset was reduced to only 6 POS tags.

- **NOM**: Non proper nominals including nouns, pronouns, adjectives, adverbs.
- **PROP**: Which is proper nouns.
- **VRB**: Active voice verbs.
- **VRB-PASS**: Passive voice verbs.
- **PRt**: Particles such as prepositions and conjunctions.
- **PNX**: Punctuation

An example of the sentence خمسون ألف سائح زَارُوا لبنَان و سوريَا في أيلول المَاضي *ḫmswn ʾlf sāʾiḥ zārwā lbnān w swryā fy ʾylwl ālmāḍy "Fifty thousand tourists visited Syria and Lebanon last September"* in the CATiB representation is given in Figure 4.12.

### 4.6.3 Comparing the three treebanks: PATB vs PADT vs CATiB

- In terms of syntactic representation PABT uses phrase structure PS and both PADT and CATiB use Dependency Structure DS. Phrase Structure is a tree representation where the words appear as leaves and the internal nodes are syntactic categories like NP (Noun Phrase) or VP (Verb Phrase). Dependency Structure, on the other hand is a tree representation except that the words are the nodes on the tree.
- In terms of syntactic structure, it is worth noting that both PADT and CATiB explicitly annotate heads and the spans implicitly; whereas PATB annotates heads implicitly and spans explicitly.

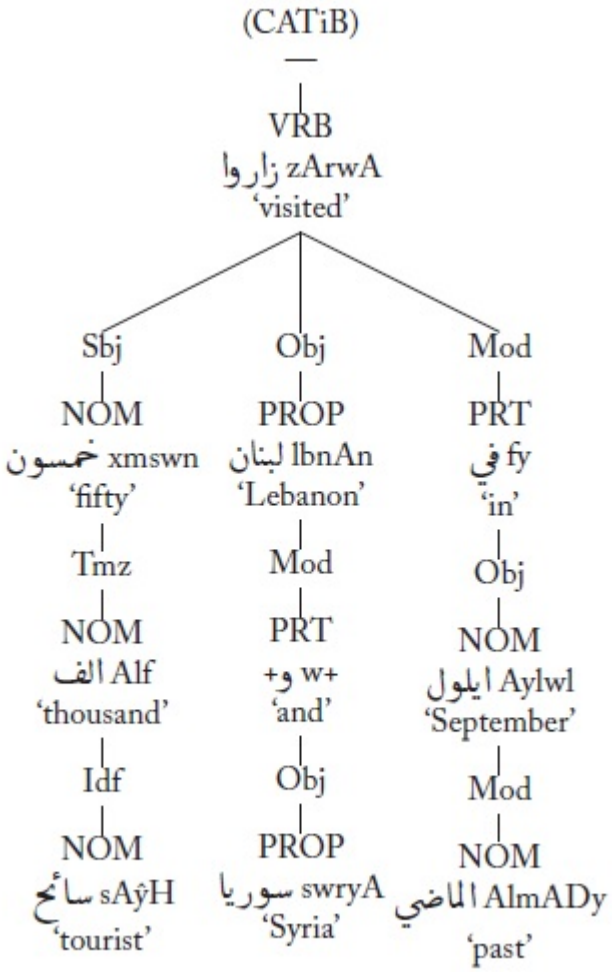



Figure 4.13: The CATiB tree of **خمسون ألف سَائح زَاروَا لبنَان و سوريَا في أيلول المَاضي**
*hmswn ʾlf sāʾiḥ zārwā lbnān w swryā fy ʾylwl ālmāḍy "Fifty thousand tourists visited Syria and Lebanon last September"* [Hab10]

- Empty pronouns are explicitly annotated in PATB, whereas in PADT and CATiB, this can be assumed to pro drop (implicit annotation).

- Word morphology: all PATB, PADT and CATiB use the same tokenisation scheme. However for POS, PATB uses over 400 tags that covers every morphological feature of Arabic words such as definiteness, case, gender, number, person, mood and voice. PADT morphology is also highly complex, for instance it makes more distinctions between nominal and adjectival definiteness, number and gender. CATiB, on the other hand only uses 6 POS tags.

### 4.6.4 MSA parsing

When it comes to parsing, Arabic faces similar challenges to other morphologically complex languages. There are several parsers used for parsing Arabic: Bikel parser (phrase structure) [KGM06], MALTParser (dependency structure) [NHN+07] and Stanford parser [SGM09]. All of these parsers require the presence of a treebank. An example of the parsers that do not require a treebank is the Attia's rule based parser.

#### 4.6.4.1 Attia's parser

This parser was built within the Lexical Functional Grammar LFG framework using XLE (Xerox Linguistics Environment) which is a platform used for developing large-scale grammars using LFG formalisms. This consists of a parser, transfer and generator components. XLE can handle UTF-8 file format, and therefore it is suitable for languages that have non-Latin alphabet like Arabic. This system is split into different modules or processes:

- The normalisation process takes the input text and corrects any misplaced white spaces and diacritics. It produces an output that can serve as a predictable text to the system.

- The second process is splitting the words into tokens.

- The morphological transducer provides morphological information for these tokens such as: their POS tag, and other morpho-syntactic features related to tense, aspect, voice, mood, gender, number and person. If the word is not in present in the core morphological analyser, there is a component called morphological guesser which provides estimates.

- With a combination of rules, notations and constraints, the system makes the decision about the correct parse. If a complete analysis is not found, then a partial parse is produced.

His approach to ambiguity was not to allow morphological choices to be made too early, and not to allow all valid and invalid solutions to appear which produces a large number of analyses that are maybe grammatically correct but make no sense in the language. So all verbs were manually reviewed taking into consideration the fact that intransitive verbs (with a small exceptions) do not have a passive, and verbs that denote perception or entity-specific change of state do not inflect for the imperative. Although this has covered lots of the ambiguity taking into consideration the precision, it required lots of human manual intervention.

#### 4.6.4.2 MALTParser

It stands for: Models and Algorithms for Language Technology Parser. It is a language-independent data-driven system with several transition-based parsing algorithms that derives dependency trees from a treebank which then can be used to parse new data for that language [NHN+07]. MALTParser consists of three essential components:

1. Deterministic parsing algorithms for building dependency graphs [YM03] [Niv03]
2. History-based feature models for predicting the next action of the parser [EBR92].
3. Discriminative learning to map histories to parser actions [Niv04]. Machine learning is used to derive a classifier given a set of transition sequences from a treebank.

MALTParser parsing algorithms can be divided into two families:

- **Stack-based:** including arc-eager and an arc-standard variant.
- **List-based:** including a projective and non-projective variant.

For a full description of the different algorithms used in MALTParser see [Niv08]. MALTParser can be run in two modes:

1. **Learning mode:** where the system takes as an input a dependency treebank and induces a classifier for guiding the parser, given specifications of a parsing algorithm, a feature model and a learning algorithm.

2. **Parsing mode:** with the same induced classifier, parsing algorithm and feature model that were used in the learning mode, the system can take as an input a set of sentences and produce a projective dependency graph for each sentence.

The format of the input should be represented in the MALT-TAB format, i.e each token is written in a line, where tabs separate the word form, POS tag, head, and dependency type. A blank line means a space that separates the sentences:

```
This    DT      2       SBJ
is      VBZ     0       ROOT
an      DT      4       DET
old     JJ      4       NMOD
story   NN      2       PRD
.       .       2       P
So      RB      2       PRD
is      VBZ     0       ROOT
this    DT      2       SBJ
.       .       2       P
```

However the output can be produced in MALT-TAB format or an XML format. The same sentences in MALT-TAB, in XML they would look like:

```
<sentence>
</sentence>
    <word form="This" postag="DT" head="2" deprel="SBJ"/>
    <word form="is" postag="VBZ" head="0" deprel="ROOT"/>
    <word form="an" postag="DT" head="4" deprel="DET"/>
    <word form="old" postag="JJ" head="4" deprel="NMOD"/>
    <word form="story" postag="NN" head="2" deprel="PRD"/>
    <word form="." postag="." head="2" deprel="P"/>
<sentence>
    <word form="So" postag="RB" head="2" deprel="PRD"/>
    <word form="is" postag="VBZ" head="0" deprel="ROOT"/>
    <word form="this" postag="DT" head="2" deprel="SBJ"/>
    <word form="." postag="." head="2" deprel="P"/>
</sentence>
```

### 4.6.5 Why is it different to our work?

In this thesis, we are collecting statistics and using them in a different way than the traditional statistical parsing. First, we do not require a treebank, and the statistics we collect are different from the statistics traditional parsers collect. We divide the edges into good and bad edges. The good ones are the ones that took part in the final analysis. The bad moves are the ones that lead directly away from the solution. We believe that what we are doing is a good thing to do because of the following:

- In MALTParser they are extracting their grammar from a from a hand-parsed corpus, where there was an implicit linguistic theory, so they already had a grammar.
- There are not many treebanks to choose from, therefore the variety of parse types generated by such systems is limited.
- Creating treebanks is a time consuming and costly task.
- There is a huge amount of manual efforts in building a treebank, whereas, in this thesis there is nothing done by hand.

The point of this thesis is that if one has a parser that they can trust their grammar, they can explore different ways in improving its speed. We already have a grammar and we are trying to learn how to use it and get the most of it, with no manual intervention.

# Chapter 5

# Machine Learning: Background

An important aspect of this thesis is to investigate the effects of using machine learning techniques on the speed of a rule based MSA parser. That is why a brief overview about machine learning is explored here.

## 5.1 Machine Learning

A central problem to Artificial Intelligence is to understand how animals and people can learn from experience to improve their skills and achieve new goals. Therefore Machine Learning is trying to find ways and program systems to automatically learn from experience. That is, to make a machine able to learn to do new things and to adapt to new situations. Machine Learning is recognising complex patterns and making intelligent decisions based on data. The problem is that the set of all possible decisions based on all possible inputs is a complex task to describe. That is why algorithms were developed to discover knowledge from specific data and experience given sound statistical and computational principles. The learning in this context is to take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on empirical data. The idea here is by observing a given data we can predict a useful output in new cases. [Mit77] provided a definition as: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. In general, machine learning is about learning to do better in the future based on what was experienced in the past. Figure 5.1 shows a typical learning problem.

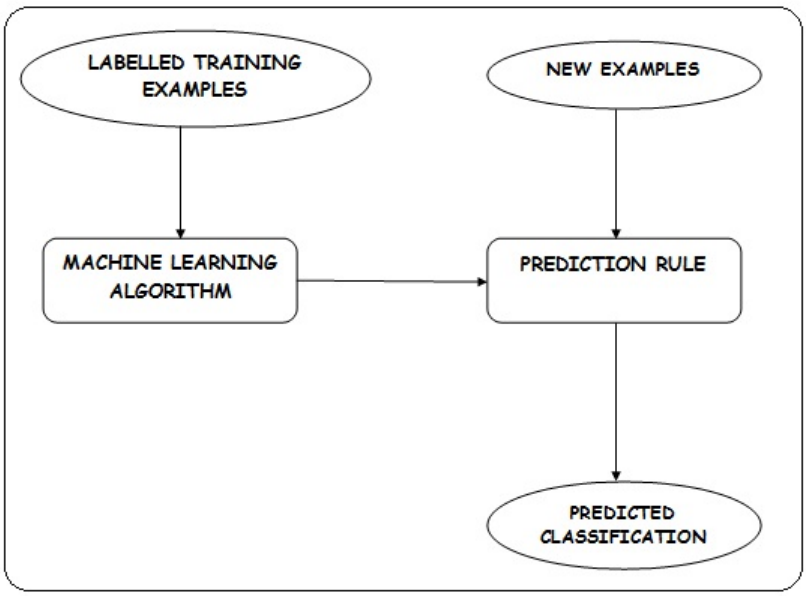



Figure 5.1: A typical learning problem [Mit77]

### 5.1.1 Concept Learning Approach

Concept learning refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. To simplify this, Mitchell [Mit77] used an example task of learning the target concept ”days on which my friend Aldo enjoys his favourite water sport. Table 5.1 describes on which day Aldo enjoys his favourite water sport or not. This determined by the last attribute *EnjoySport*. The task is to predict the value of EnjoySport for an arbitrary day.

| Day | *Sky* | *AirTemp* | *Humidity* | *Wind* | *Water* | *Forecast* | *EnjoySport* |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

Table 5.1: Positive and negative training examples for the target concept EnjoySport [Mit97]

### 5.1.2 Decision Tree Learning

Decision Tree Learning is a method that uses inductive inference to approximate a target function, which will produce discrete values. It is widely used, robust to noisy

data, and considered a practical method for learning disjunctive expressions [Mit97]. The goal of a decision tree is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. It classifies instances by sorting them down the tree from the root to the leaf nodes. For example, one that enjoys playing Tennis in a comfortable weather, specifies the uncomfortable conditions that stop him from playing Tennis in a tree as in Figure 5.2.

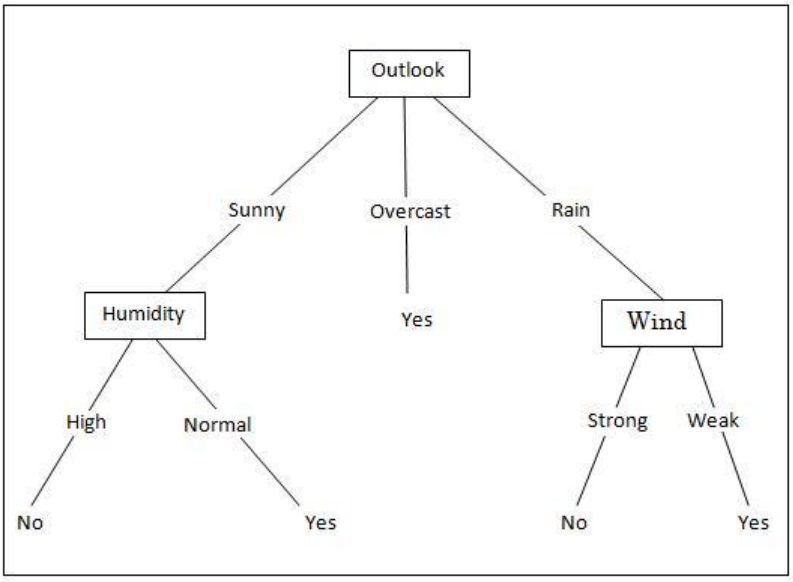



Figure 5.2: A decision tree for the concept playTennis [Mit97]

Decision trees can be represented in tables or as a set of boolean expressions (Disjunctions of conjunctions):

$$(Outlook = Sunny \wedge Humidity = Normal$$
$$\vee\ Outlook = Overcast$$
$$\vee\ Outlook = rain \wedge Wind = Weak)$$

Decision trees are best suited to problems with the following characteristics:

- Instances that are described by attribute-value pairs. For example an attribute *Temperature* which has three values *(hot, mild, cold)*.
- The target function has discrete output values. The decision tree in Figure 5.2 assigns a classification *yes, can play sport* or *no, cannot play sport*.
- Disjunctive hypothesis may be required.

- The data might possibly be noisy, i.e. it may contain errors.
- The training data may contain missing attribute values. Decision tree methods can be used even when some training examples have unknown values.

### 5.1.3 Decision Tree Algorithms

Having discussed the decision tree technique, the next step was to select the most appropriate algorithm for our experiments. We are learning facts about edges which has a lot of missing values. Therefore, the chosen algorithm has to deal adequately with this missing data. The algorithm that satisfies this requirement was J48, which is a Java implementation of the well known C4.5 decision tree algorithm. We chose J48 mainly because C4.5 is one of the best known and most widely used learning algorithms. Also because J48 handles the missing values in the data. C4.5 was preceded by ID3 and produces a comprehensible classification model in a decision tree form. The implementation of J48 was produced through WEKA (Waikato Environment for Knowledge Analysis) which is a popular suite of machine learning software developed at the University of Waikato. Below is a brief discussion of ID3 and C4.5.

- **ID3**: which stands for Iterative Dichotomiser 3, is an algorithm used to generate decision trees invented by Ross Quinlan [Qui92]. It constructs decision tree by employing a top-down, greedy search through the given sets of training data to test each attribute at every node. ID3 uses a statistical property called *Information Gain* to decide which attribute is the best. Information Gain measures how well a given attribute separates training examples into targeted classes. The best attribute is the one that provides the greatest information gain. This best attribute is placed as a root node of the tree. Repeating this process on each partition of the divided data, will create sub trees until the training data is divided into subsets of the same class. Entropy is used to determine which node to split next in the algorithm. It is computed as follows [Mit77]:

$$E(S) = -\sum_{j=1}^{n} f_S(j) \log_2 f_S(j) \tag{5.1}$$

$E(S)$ is the information entropy of the set S.
$n$ is the number of different values of the attribute in S.
$f_S(j)$ is the frequency of the value j in the set S.
$\log_2$ is the binary logarithm.

For example, if we are given $S$ as a set of 16 examples with 10 positive examples, and 6 negative examples, then the entropy is calculated as follows:

$$E(S) = -(10/16)\log_2(10/16) - (6/16)\log_2(6/16) = 0.954$$

From this equation we can notice that entropy will be 0 if all members of $S$ belong to the same class. When this happens, the data is perfectly classified.

After computing the entropy we can now compute the information gain to estimate the gain produced when splitting the data over an attribute:

$$G(S,A) = E(S) - \sum_{i=1}^{m} f_S(A_i)E(S_{A_i}) \tag{5.2}$$

$G(S,A)$ is the gain of the set S after splitting over the A attribute.
$E(S)$ is the information entropy of the set S.
$m$ is the number of different values of the attribute A in S.
$f_S(A_i)$ is the frequency of the items possessing $A_i$ as value for A in S.
$A_i$ is $i^{th}$ possible value of A.
$S_{A_i}$ is a subset of S containing all items where the value of A is $A_i$.

for instance, let us consider a sample set $S$ of 16 examples in which one of the attributes is wind speed (with values *Weak/Strong*). Classifying these resulted in 10 positive and 6 negative examples. Assuming that there are 9 occurrences of $Wind = Weak$ and 7 occurrences of $Wind = Strong$. For $Wind = Weak$ there are 6 positive and 3 negative examples. For $Wind = Strong$ there are 4 positive and 3 negative examples. Computing the gain $G(S,Wind)$ is as follows:

$$Entropy(S_{Weak}) = -(6/9)\log_2(6/9) - (3/9)\log_2(3/9) = 0.918$$

$$Entropy(S_{Strong}) = -(4/7)\log_2(4/7) - (3/7)\log_2(3/7) = 0.985$$

$$Gain(S,Wind) = Entropy(S) - (9/16) \times Entropy(S_{Weak}) - (7/16) \times Entropy(S_{Strong})$$

$$Gain(S, Wind) = 0.954 - (9/16) \times 0.918 - (7/16) \times 0.985 = 0.007$$

These calculations are done for every attribute to calculate the information gain and the highest gain is the one used in the decision node.

- **C4.5**: One trouble of ID3 is that it is overly sensitive to features with large numbers of values. Also Id3 of WEKA only runs on nominal attributes. C4.5 is an extension of the earlier ID3 algorithm. It builds decision trees from a set of training data in the same way as ID3 does using entropy concept. There are some improvements in C4.5 over ID3, some of them are:
  - Handling both continuous and discrete attributes.
  - Handling training data with missing attribute values.
  - Pruning trees after creation. Once the tree is created the algorithm attempts to remove branches that do not help by replacing them with leaf nodes.

### 5.1.4 Learning from solution paths

Our approach involves learning from moves that lead to the final analysis, and from moves that lead away from it. This idea was first employed by Mitchell [SLM82] in his LEX system. He discussed learning from domains where search was involved. He introduced his system LEX which is a computer program that acquires problem solving heuristics in the domain of symbolic integration. LEX operates in the domain of symbolic integration in which it solves integration problems by searching through a space of mathematical expressions containing indefinite integrals. The operators for traversing the search space are standard integration rewriting rules. If the problem state contains a subexpression of one of the left hand side of the rule, then this will be replaced by the right hand side, for example: $\int r.f(x)dx \rightarrow r \int f(x)dx$ which means if the problem state contains a subexpression of the form $\int r.f(x)dx$ (where $r$ is any real number, and $f$ is a function), then that subexpression can be rewritten with the real number outside the integral. The problem solving goal is to derive a problem state that contains no integrals.

LEX learns heuristics (conditions under which operators should be applied). It assigned credit when a correct operator was used and assigned a blame when an incorrect operator was selected. Therefore, there were two types of instances:

1. Positive instances: They are the steps that lie on the solution path.

2. Negative instances: they are the search steps that lead from some state on the solution path to some state off the solution path.

The idea of Mitchell was that once a solution path has been found, this can be used to assign credit and blame to instances of these operators. On the one hand any move that leads from a state on the solution path to another state on the solution path, is labelled as a positive instance. On the other hand, A move that leads from a state on the solution path to a state not on the path is labelled as a negative instance.

In this thesis, we are investigating a similar idea to Mitchell learning from solution paths. The first step is to learn to distinguish desirable from undesirable moves, and to determine what features are responsible for these moves. This approach involves parsing a set of sentences and learn from all the solution paths. Moves leading to states on the solution path are good moves (desirable), since they lead to a solution, while moves directly going off the path are bad moves (undesirable). The third type of moves are neutral, they are moves that follow on from bad moves, since they are not responsible for going off the solution path, and the parser should never have reached these states in the first place.

# Chapter 6

# Optimising the rule based parser

In order to discuss the optimisation of the parser, let us first introduce the details of the syntactic framework. The general syntactic framework lies somewhere between *HPSG* and pure categorial grammar, but with a rather radical approach to extraposition.

## 6.1 Head-Driven Phrase Structure Grammar *(HPSG)*

A considerable amount of work have been done on linguistic theories based on constraints. The notion of constraints is generally used to represent properties that an object must satisfy. Most linguistic theories currently use this approach, particularly constraint-based approaches. This includes the three theories: Lexical Functional Grammars *(LFG)*, Generalised Phrase Structure Grammars *(GPSG)*, and the most recent and widely accepted *HPSG*. The grammar used in this thesis has a great deal in common with *HPSG*. Therefore, an introduction to *HPSG* is presented here.
HPSG is a constraint-based approach to grammatical theory, which was developed as a non-derivational grammatical framework. Non-derivational means that *HPSG* has no notion of deriving one structure from another one. Instead, different representations are just sub-parts (features) of a single larger structure related by declarative constraints of the grammar [Pol]. *HPSG* is also described as highly lexicalist in that it makes use of a rich and complex lexicon in its representations. These representations use feature structures, often written as attribute-value-matrixes (AVM), to represent grammar principles, grammar rules and lexical entries. A fundamental concept in *HPSG* representations is that of a *sign*, which is a collection of properties or information, including phonological, syntactic, semantic, and contextual constraints. Both words and phrases are represented as signs. All signs must have at least two attributes: PHONOLOGY

(abbreviated to PHON) whose values represent a list of phonological descriptions, and *SYNSEM* for syntactic and semantic constraints. Phrase signs specify the attribute *DTRS* (for DAUGHTERS) which represents the constituent structure of the phrase. As an illustration, a partial *HPSG* lexical sign for the word *put* may look like:

$$\begin{bmatrix} \textit{word} & & & & \\ \text{PHON} & & \langle \textit{put} \rangle & & \\ \text{SYNSEM} & \text{LOCAL} & \text{CAT} & \text{HEAD} & \begin{bmatrix} \textit{verb} & \\ \text{AUX} & \textit{non} \end{bmatrix} \\ & & & \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle \text{NP}\boxed{1} \rangle \\ \text{COMPS} & \langle \text{NP}\boxed{2}\ \text{PP}\boxed{3} \rangle \end{bmatrix} \\ & & \text{CONT} & \begin{bmatrix} \textit{put-relation} & \\ \text{PUTTER} & \boxed{1} \\ \text{THING-PUT} & \boxed{2} \\ \text{DESTINATION} & \boxed{3} \end{bmatrix} & \end{bmatrix}$$

An AVM *sign* consists of attributes (written in capital letters), and their values which can be atomic such as *verb* or complex as nested AVMs. In the above example, *PHON* (phonological description) is represented as a list with the word *put*. *SYNSEM* (syntactic and semantic constraints) is divided into LOCAL and NONLOCAL. As this is only a partial representation only LOCAL is shown. LOCAL is in turn divided into CAT (category), and CONT (content). CAT holds syntactic information about the sign and is divided into HEAD and VAL (valence). The HEAD value is generally thought of as syntactic category which contains information about the sign itself, i.e. in this example the word *put* is a non auxiliary verb. VAL states what arguments the word *put* takes. CONT contains the semantic relation for the word *put*, and the roles of its arguments.

## 6.2 The grammatical framework

In this thesis, we use an *HPSG-like* grammar [Ram99], where an individual sign contains information about the structure and appearance of a given phrase in addition to the usual phonology and syn-sem. The parsing algorithm was adapted to work with

free word order languages, especially the treatment of *extraposition* which will be discussed later.

## 6.3 Chart parsing

Because we are dealing with a very ambiguous language (MSA) a chart parser is the choice. A chart consists of a collection of *vertices*, one between each word of the input, connected by *edges*, labelled with grammatical information as shown in Figure 6.1:

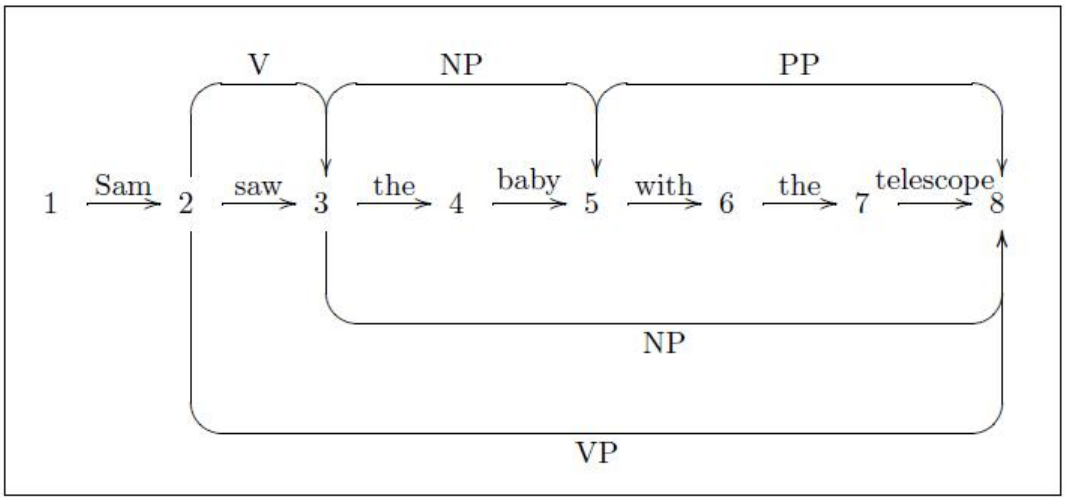



Figure 6.1: An example of a chart [Arn]

In general the advantages of a chart parser can be summarised as:

- Avoiding duplication where everything is represented but only once. The idea here is that before trying to parse any part of the input as a PP for example, we look in the chart and check if it was parsed as a PP before.
- Providing a local representation for local ambiguity.
- Providing a representation for partial parses.

## 6.4 Active Chart parsing

A passive chart is used to keep a record of complete constituents that we have found, whereas in an active chart, we record what it is we are actually looking for and how much of it we have found so far. An active chart consists of :

- A chart which may contain incomplete edges.
- An agenda which shows the order in which tasks are carried out.

Dotted rules are used to represent stages in the parsing process. The rule: $NP \rightarrow DET\ N\ PP$ corresponds to the following rules:

1. $NP \rightarrow \circ\ DET\ N\ PP$
2. $NP \rightarrow DET \circ N\ PP$
3. $NP \rightarrow DET\ N \circ PP$
4. $NP \rightarrow DET\ N\ PP \circ$

For example rule 3 means part of an NP has been found (the DET and the N), and a PP remains to be found to complete the NP. In order to see how the parser in this thesis provides a way of dealing with free phrase order, we will first review the standard version of chart parsing. The main benefit of using chart parsing is to avoid parsing items that have already been parsed, and to keep track of all phrases and sub-phrases that have been considered. The Fundamental Rule of Chart Parsing (FRCP) is: if X and Y are adjacent structures, where X is unsaturated and adding Y to it will saturate it, then X and Y can be merged to form a saturated structure. The following example explains this principle:

1. 1, 2, DET, [], [], ...
2. 2, 3, N, [], [], ...
3. 1, 2, NP, [DET], [N], ...
4. 1, 2, NP, [DET], [N, PP], ...

rule 3 and rule 2 can be merged together to make:
1, 3, NP, [DET, N], [], ...

## 6.5 Refining the Fundamental Rule of Chart Parsing

In a sentence like ”Jack and Jill went up the hill”, the implementation of the sign holding information of the word ”and” may look something like this:

```
{sign(structure(positions(start(1), end(2), text(and))),
    syn(nonfoot(head(cat(U), ...)),
        subcat(args([sign(structure(dir(+right, -left)),
            syn(nonfoot(head(cat(U), ...),
            meaning(...)),
            sign(structure(dir(-right, +left)),
            syn(nonfoot(head(cat(U), ...),
            meaning(...)),
        foot(slash=defval(value([]), EM),
            focus=defval(value([]), EN),
            wh=defval(value([]), EO))),
    meaning(...))}
```

A lot of information can be embedded within a sign. Two important ones here are: positional feature, and arguments. The positional information is explicitly encoded inside the sign (the word ”and” start at position 1 and end at position 2). ”and” is expecting two arguments: first one should appear to the right and the second one should appear to the left. ”and” with both arguments should form an item of the same type as the arguments.

The FRCP allows appropriate items to be combined only if their start and end points are identical. In order to cope with situations where items can be displaced, most grammatical theories introduce some sort of empty items. If an item is thought to be expecting an argument, and this argument is not there, a trace can be introduced. When finding an item that does not appear to be doing anything useful where it is, and can fill in the empty item, it is assumed that it is the wanted argument.

To allow this to happen the FRCP needs to be refined and assume that: if X is a sentence that has a trace within itself and Y is an adjacent item of the same type, then they can be merged to form a sentence which is not missing anything. The problem of this theory is that empty traces are introduced everywhere, which can result in a huge number of edges to be considered. In addition if the wanted argument is not found then this trace will never be canceled. To improve this theory and avoid introducing traces everywhere, it was suggested by Johnson and Kay [JK94] cited by [Ram99]) that a trace can be inserted only if you find a sponsor for it. However, the disadvantage of this theory is that you allow items to be sponsors, then they turn out to be irrelevant. It was proposed that you can make a search for sponsors, but this makes the search for sponsors more expensive. Ramsay’s approach to situations where items can appear out

of positions is retrieving those items directly rather than introducing empty items and sponsors. Instead of using sponsors to guide us whether we can introduce empty traces or not, why not just use this sponsor without introducing empty items? To make this work, the FRCP should be revised to:
if X and Y are non-overlapping structures, where X is unsaturated and adding Y to it will saturate it, then X and Y can be merged to form a saturated structure covering the combined spans of X and Y. Here is the explanation of the notion ”overlapping”: a span of an edge is represented by a bit-string as shown in Figure 6.2. For instance: ”I wear clothes which she bought”, the edges corresponding to the single ’I’, ’wear’, ’clothes’, ’which’, ’she’ and ’bought’ would have spans 1, 10, 100, 1000, 10000, 100000 respectively.

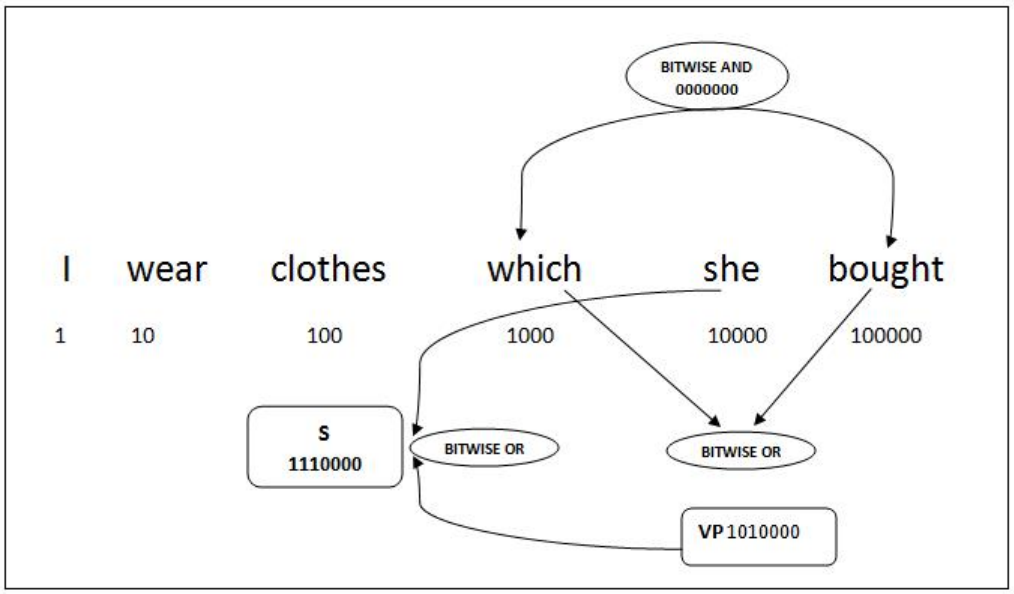



Figure 6.2: An example of overlapping items using bitwise ”AND” and ”OR”

We use bitwise ”AND” to check whether two items are overlapping or not. if X AND Y gives 0, they are non-overlapping. bitwise ”OR” is used to combine two items. In the example in Figure 6.2 ”bought” and ”which” can be combined because $1000000 \wedge 10000$ gives 0. The span of the combination is $1000000 \vee 10000$ which gives 1010000.
In order to define constraints on extraposition, some features are introduced here, mainly: compact, xstart, and xend. A phrase is compact if it contains all the words that appear between its extreme left and right ends. It is easy to check whether a phrase is compact or not. If I is the extreme start of a phrase and J is its extreme end,

and SPAN is the bit string encoding its span then the phrase is compact if and only if SPAN equals ((1 ≪ (J-I))-1) ≪ I. If we consider the example: ”who he gave it to”. The phrase ”gave it” consists of a verb ”gave” that is attached to its direct object ”it” to form a compact phrase. The resulting sign would look like:

```
sign(structure(positions(start(2),
                         end(4),
                         span(01100),
                         +compact,
                         xstart(2),
                         xend(4))),
    syn(nonfoot(head(cat([xbar(+v, -n)]), ...), ...),
        subcat(args(["PP", "NP"]))),
    meaning(...))
```

Checking whether this phrase is compact or not is described in Figure 6.3.

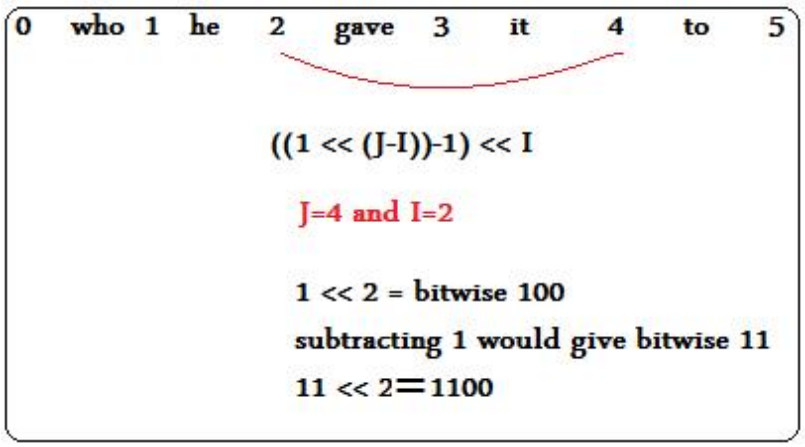



Figure 6.3: Checking whether a phrase is compact

We can note here that a phrase is compact if simply its extreme start is the same as its natural start, and also its extreme end is the same as its natural end. For example the preposition ”to” will combine with ”who” to form a non-compact phrase whose extreme start is 0 which is the start of ”to”, and whose natural start is 4 which is the start of the head. The extreme and natural ends are both 5. So the sign would look like:

```
sign(structure(positions(start(4),
                         end(5),
                         span(10001),
                        -compact,
                         xstart(0),
                         xend(5))),
    syn(nonfoot(head(cat(pp), ...), ...),
        subcat(args([]))),
    meaning(...))
```

So far we have discussed the syntactic framework and the parsing algorithm that dealt with situations where items can appear out of their canonical position. The next section will discuss the lexicon and its limitations.

## 6.6 Improving the parser and extending the dictionary

The parser has got a fairly small dictionary (about 295 words). Therefore building a large lexicon was necessary to run these experiments. As most words are built up from roots by following certain patterns and adding infixes, prefixes and suffixes, there are large numbers of words that comprise semantically related variants of a common core. If we consider the root كتب *ktb*, adding the diacritics, infixes, prefixes and suffixes, we get various aspects of writing: كُتُب *kutub* (books), كِتَاب *kitāb* (book), كَتَبَ *kataba* (wrote), كَتَّبَ *kattaba* (made write), مَكتَبَة *maktabh* (library), كِتَابة *kitābh* (writing), مَكتَب *maktab* (desk), كَاتِب *kātib* (writer), كُتَّاب *kuttāb* (writers),...etc. Each of these words share the 3 consonants كتب *ktb* and they even have related meaning which is something to do with writing.
The question was whether we should store all the inflected forms separately in the lexicon or only store the root, and add more information to derive the inflected forms including derivational affixes and diacritics. In this lexicon, all the affixes that can attach to a root are listed explicitly. Words are stored as variations on a single lexical entry together with information about what derivational affixes can be attached to it, as well as the diacritics that are associated with each affix.

### 6.6.1 Nouns

Nouns are stored as roots and the affixes that can be attached to it are specified. For example, if we consider the root درس *drs* can accept the derivational prefix مُ *mu* and the diacritics *a*, and *i*, in addition to the sound ر *r* being doubled to make the noun مُدَرِّس *mudarris*. The noun entries in this thesis are represented in a prolog form as follows:

```
"k?t?b" lextype regular(nominal,
       [’’:[["i", "A"]:[_]:broken(’’):book:[masculine, translation(book)],
    ["u", "u"]:[_]:broken(’’):book:[third_plural_only, translation(book)],
    ["i", "A"]:[_]:regular: ~living:[feminine, translation(writing)],
    ["a", "y"]:[_]:regular: ~living:[singular, feminine, translation(army1)],
    ["A", "i"]:[_]:regular:human:[translation(writer)],
    ["u", "A"]:[C2,C2]:broken(’’):human:[third_plural_only, translation(writers)]
   ],
’I’:[["ti", "A"]:[_]:regular: ~living:[]],
’’’isti’:[["o", "A"]:[_]:broken(’’):thing:[not_plural,masculine,translation(dictation)]
  ["o", "A"]:[_]:broken(’’):thing:[third_plural_only,translation(dictation)]],
{m,a}:[["o", "w"]:[_]:regular: ~living:[], % written (?)
       ["o", "a"]:[_]:regular: ~living:[masculine, translation(maktab)],
       ["o", "a"]:[_]:regular: ~living:[feminine, translation(maktabp)]       ]
       ], 1)
       ::: noun delayed ntype(simpleArabic).
```

The characters between the square brackets replace the question marks to make a diacriticised word. For example: the entry ”k?t?b” and [”u”,”u”] will result in the diacriticised word ”kutub”.

### 6.6.2 Verbs

Most MSA verbs consist of three consonants and a diacritic pattern specifying what goes in between the consonants, and also a pattern that determines whether one of the root consonants should be duplicated. This information is stored so that there is an interaction between the stem and the affixes that can attach to it. For instance, the form of the present prefix depends on the specific stem it is attached to and also on the verb being either active or passive. There are 4 derived forms, for example the two roots كتب *ktb* and كتّب *kttb*:

1. Active present → يَكتُب *yaktub* and يُكَتِّب *yukattib*
2. Active past → كَتَبَ *kataba* and كَتَّبَ *kattaba*
3. Passive present → يُكتَب *yuktab* and يُكَتَّب *yukattab*
4. Passive past → كُتِبَ *kutiba* and كُتِّبَ *kuttiba*

The two active derived stems كتُب *ktub* and كَتِّب *kattib* take their third singular prefixes as يَ *ya* and يُ *yu* respectively. The passive forms of the same verbs كتَب *ktab* and كَتَّب *kattab* both take the present prefix يُ *yu*.

### 6.6.3 The challenge

In the absence of MSA lexicons, the challenge was to automatically create a fairly big dictionary that at least can be used to run the experiments. Extending the dictionary revealed a number of weaknesses in the morphological analysis in the original system. This include the presence of weak letters (و *w*, ي *y*, ى *ā*, ا *ā*) in verbs. Arabic triliteral verbs are divided into:

- Strong verbs: They are formed by the pattern CVCVC (C refers to the consonants and V refers to short and long vowels). Strong verbs have three types:
  - regular: when there is no أ *ʔa*, the shadda '∼' (doubling the consonant), or weak letters. For instance: درس *drs* "to learn"
  - Hamzated: One of the radicals is a hamza أ *ʔa*, for example أَكَل *ʔakal* "to eat".
  - Doubled: One of the radicals is doubled for example: رَدَّ *radda* "to reply".
- Weak verbs: When one of the radicals is a weak letter (و *w*, ي *y*, ى *ā*, ا *ā*). Depending on the position of the weak letter This is divided into:
  - Assimilated: The first radical is either و *w* or ي *y*, e.g. وَجَد *waǧad* "to find".
  - Hollow: The middle radical is a weak letter, e.g. قَالَ *qāla* "to say".

- Defective: The last radical is either ا *ā* or ى *ā*, e.g. شكَى *škā* "to complain".
- tangled: Both the first and third are weak letters, e.g. وَفَى *wafaā* , or the second and third radicals are weak letter, e.g. رَوَى *rawaā*.

The problem with these weak verbs is that they do not follow the same patterns a regular verb would follow as shown in Table 6.1:

| Past tense | Present tense | Imperative |
|---|---|---|
| كَتَبَ *kataba* (This is regular) | يَكتُب *yaktub* | أُكتُب *ʾuktub* |
| وَقَفَ *waqafa* (remove و *w*) | يَقِف *yaqif* | قِف *qif* |
| قَالَ *qaāla* (convert ا *ā* to و *w*) | يَقُول *yaquwl* | قُل *qul* |
| رَمَى *ramaā* (convert ى *ā* to ي *y*) | يَرمي *yarmy* | إرمي *ʾirmy* |
| وَفَى *wafaā* (remove و *w*) | يَفي *yafy* | فِ *fi* |

Table 6.1: An example of the irregularity of weak verbs

# Chapter 7

# Integrating the POS tagger and edge classifier into PARASITE

This thesis is concerned with running a variety of experiments on the original rule based parser, and investigating their effects on the speed of this parser. These experiments are divided into three main parts:

- Investigating the effects of using a POS tagger with the original parser.
- Investigating the effects of using machine learning and WEKA on the speed of the original parser.
- Investigating the effects of using both the POS tagger and Machine learning at the same time on the speed of the original parser.

In order to illustrate how the POS tagger was integrated into the parser and how the classifier was used, let us first describe the system before applying the experiments. The system used in this thesis is a Sicstus prolog program used to perform a range of linguistics tasks including parsing. It is a chart parser where all the information we know about a word is stored in a sign. The following shows the form of part of a sign of the word كتَابة *ktābh* (writing):

```
{'syntax(nonfoot(head(cat(xbar(-v, +n)),
    agree(person(first(num(-singular, -dual, -plural)),
            second(num(-singular, -dual, -plural)),
            third(num(+singular, -dual, -plural))),
        count(_A, _B, _C, generic(def(-value, usedef(_D)))),
```

```
        gender(-masculine, +feminine, -neuter),
        _E),
    nform(case(-caseDefault, -voc, -pcase), -date)),
    mcopy(type([noun, -pronoun | _F]), -bracketed),
.
.
.
-clitic))),
meaning(semantics(noun('k?t?b'))),
remarks(translation(writing), sense(ktbn2), score([20 | _G]))'}'
```

This sign holds all the information we know about the word كتَابة *ktābh*, in particular, +feminine, +singular, +n means it is a feminine singular noun. The last line which is the remarks feature contains an important feature called score which assigns a score to all entries in the agenda. In this example the word كتَابة *ktābh* was assigned a score of 20. The lower the score the better. The parser is a chart parser, with edges asserted in the Prolog database, where the score is used for deciding which edge should be extended next. The score of the resulting word can be computed in different ways such as:

1. Choosing the minimum of the individual edges.
2. Choosing the maximum of the individual edges.
3. Just adding both scores.
4. Computing the average of the individual scores.

The original parser simply added the scores when combining the edges. Therefore choosing different ways of combining weighting factors is a potentially useful thing to do. The first two above are not good for this purpose, since combining one very good with one very bad will not give an accurate score. We found out the one that gives good results is computing the average. It was noted that combining scores by using the average of the penalties associated with the edges being combined is substantially better than adding them as will be shown in the next chapter.

Figure 7.1 shows the overall architectural chart of the system, which can be summarised as follows:

- The system takes a sentence as an input.
- The next step is pre-processing which is about manipulating the text before even starting looking in the dictionary. The things included in this step are dealing with elisions such as can't and isn't, dealing with numbers and spotting if the commas and dots used are part of the number.
- Lexical lookup: Here the surface forms that we get from the pre-processing are looked up in the dictionary. Words may be known to have good or bad properties which are expressed in terms of scores.
- In this step clitics and affixes are separated from the main stem. This is a significant step for Arabic because it includes dealing with the definite article ال *āl* "Al" as well as clitic pronouns, prepositions and conjunctions.
- Parsing and combining fragments together. In this step we note that individual edges have scores. The process of combining two edges may contribute to the score for instance agreement failure, marked order, or zero subject.

## 7.1 Integration of the tagger into the parser

As shown in Figure 7.2 words are passed to the tagger to be tagged. The tag assigned by the tagger is compared to the tag assigned by the parser. A good score is given if they agree, and a penalty is applied if they disagree.

```
scoreTags(PTAG, NTAGS, WORD) :-
    (tagPenalty(TPEN) ->
     (cmember(PTAG:N, NTAGS) ->
      J is -1*TPEN*N;
      J = TPEN);
     true),
     penalise1(WORD, J).
```

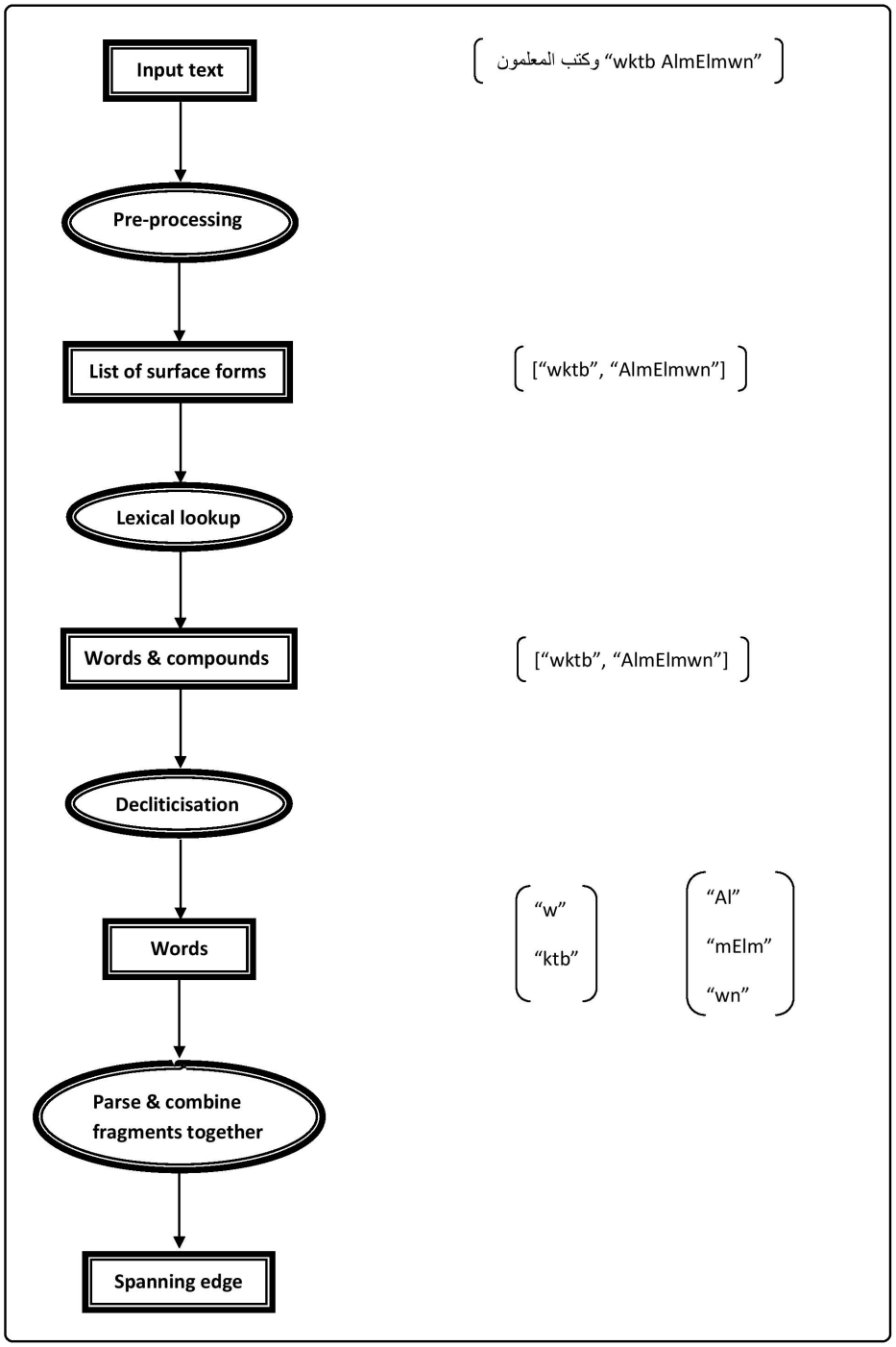



Figure 7.1: The architecture of the original system

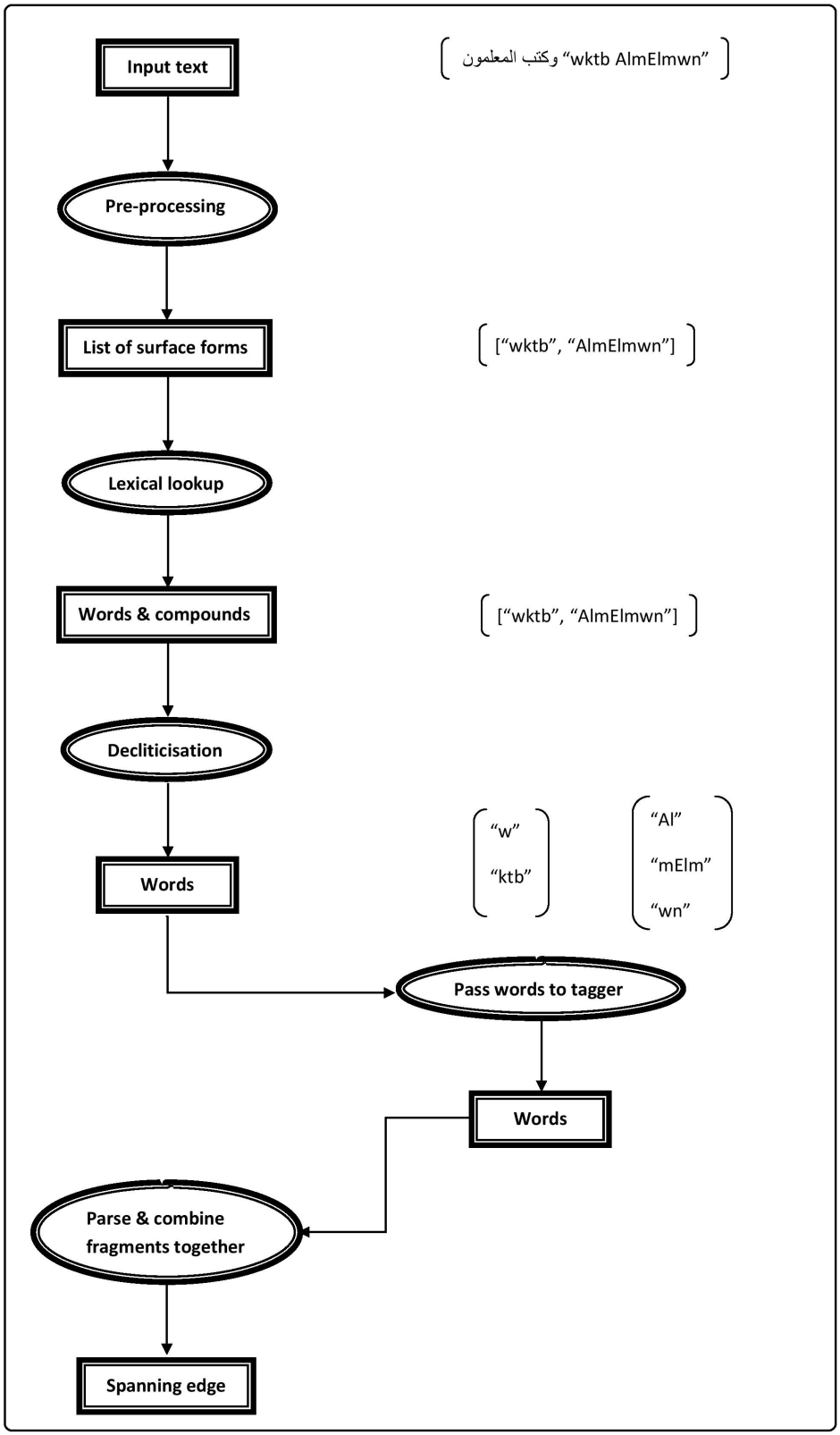



Figure 7.2: A modified architecture of the system after using the tagger

## 7.2 Integration of the edge classifier into the parser

WEKA (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato [HFH+09]. WEKA is a free software available online. It is a collection of machine learning algorithms. It contains tools for data preprocessing, classification, regression, clustering, association rules, and visualization. After parsing the training set, the facts about edges were stored in an ARFF format (Attribute-Relation File Format), since WEKA reads this type of files. An ARFF file consists of a header that describes the attribute types, and a data section which is a comma separated list of data. An example of an ARFF is shown in Appendix B.

In this thesis, ARFF files were used to store facts about edges and classify their status whether good, bad or neutral. For this purpose J48 was selected as classifier. Edges that appear on the actual solution path are classified as good edges, edges that lead directly away from the solution path are considered to be bad edges, and edges that follow on from bad edges, they are considered to be neutral because it is their parents fault that diverted away from the solution path. For this purpose, if an edge is classified good, a good score is given. However, if an edge is bad then a penalty is applied. For example, the code below shows the way we assign these penalties. Smaller penalties are better. As will be shown in the next chapter, the range of penalty N is from -200 to 0. Therefore, if a move is classified good, it will be assigned a penalty of N. However, if it is classified as bad, it will have a penalty of -N as shown in the code below.

```
goodOrBad(Z) :-
    (\+ \+ (call_residue(getBaseARFF(Z, ARFF), _),
       tryj48(ARFF, STATUS)) ->
(useJ48(N) ->
    ((STATUS = good) ->
penalise1(Z, N);
(STATUS = bad) ->
penalise1(Z, -1*N);
true);
    true);
true).
```

# Chapter 8

# The experiments and results

In this chapter we explore the effects of using the tagger only, the WEKA classifier only, and both the tagger and the classifier at the same time. We also investigate the effects of these on the accuracy of the parser.
In order to run the experiments, we created a corpus of Arabic sentences. By parsing these sentences , we are interested in the time it takes and the number of edges it makes. The settings for this experiment is shown in Table 8.1.
The sentences used were extracted from several Arabic websites of children stories. To get the sentences, the text was split on the full stop. As the text was written in Arabic script, the sentences were then transliterated using the Buckwalter Transliteration. The reason why we chose Buckwalter is simply because every Arabic character is represented by one and only one Latin character. A sample of these sentences is attached in Appendix C.

| | |
|---|---|
| Size of the training corpus | 800 sentences |
| Size of the testing corpus | 200 sentences |
| Source of the corpus | Children stories from different web sites online |
| Metrics | Number of edges and Time(Millisecond Ms) |

Table 8.1: Experiments settings

In order to investigate the effects of using the tagger and machine learning on the speed of the parser, one will measure the time taken to parse a sentence. When measuring the time, there are some fluctuations in the timing that are due to other things happening in the computer at that time which are unpredictable. Therefore counting the number of edges is something we can do very precisely. As we will see later on,

there is a strong correlation between the number of edges and the time taken to parse a sentence. That is why, the number of edges is considered to be a surrogate for time in a reasonably reliable way. If there are fewer edges, one would expect the time to be less.

## 8.1 Integrating the tagger

This section describes integrating a part of speech tagger into the parser and investigating its effectiveness on the speed of the parser. The biggest problem one would face when parsing MSA is ambiguity, which leads to a very large number of analyses being produced, for instance a small sentence like: كتب درس *ktb drs*, whose most plausible translation is "He wrote a lesson", will produce around 19 edges. The word كتب *ktb* can be كَتَبَ *kataba*, كَتَّبَ *kattaba*, كُتِبَ *kutiba*, كُتِّبَ *kuttiba* which are verbs or كُتُب *kutub* which is a noun, and the word درس *drs* can produce دَرَسَ *darasa*, دَرَّسَ *darrasa*, دُرِسَ *durisa*, دُرِّسَ *durrisa* as verbs, or دَرس *dars* as a noun. This technique has worked well for English, and here we are looking at its effects.

This experiment includes parsing a test set of 200 sentences in two ways:

- Parsing the test set without the use of the tagger.
- Using the tagger to guide the parser by introducing penalties (scores). A good score is given if the tagger and parser agrees on the tag. However if they have different tags, then a bad score is given. Here we give a weight to tagger's suggestion which encourages the edges that the tagger suggests.

### 8.1.1 The results

The average time taken to parse a sentence before integrating the tagger is 1186.23 ms. However, the same experiment and integrating the tagger resulted in 1065.90 ms as the average time per sentence.

### 8.1.2 Analysis

Integrating the tagger resulted in an improvement from 1186.23 ms to 1065.90 ms, approximately 10% improvement on time taken per sentence. By analysing the trees in both tests, it is worth noting that only 2 out of the 200 that parser succeeded in

producing an analysis without the tagger, but failed when using the tagger. Also, on one hand without using the tagger, the parser produced 1950 good edges, 5152 bad edges and 9897 neutral edges. On the other hand when using the tagger, it constructed 1885 edges, 4766 bad edges and 9022 neutral edges, as shown in Figure 8.1.

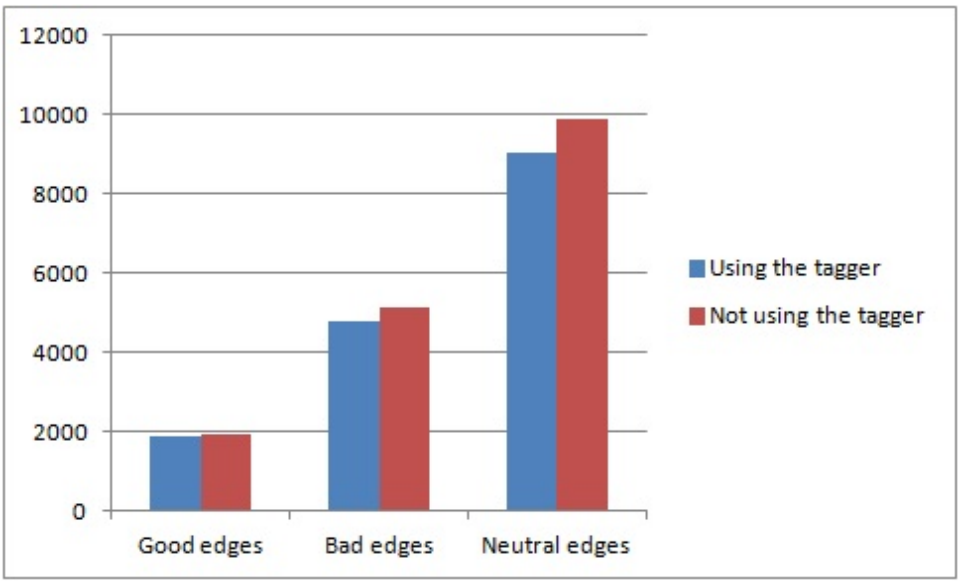



Figure 8.1: The number of edges created by the parser: Using the tagger vs Not using the tagger

## 8.2 Experiments using different parameters

We run a set of experiments with different parameter settings as arguments. These parameters are whether we use hand-coded penalties (score) or not, whether we use tagging or not, and the type of combining the scores (average or sum). These combinations are described in the following experiments:

1. Using Hand-coded penalties, no tagging, combine penalties by averaging them:

```
runTest((retractall(soft(_, _)),
        assert(soft(_, 77)),
        retractall(bound(_)),
        assert(bound(1000)),
        unset(ignorePenalties),
        unset(sumScores),
```

```
       retractall(tagPenalty(_)),
       set(unknownWords)),
      arabicTest).


GOOD 2008, BAD 5536, NEITHER 11105
Total number of edges used: 18649
```

2. Using hand-coded penalties, no tagging, combining penalties by addition:

```
runNassimTest((retractall(soft(_, _)),
       assert(soft(_, 77)),
       retractall(bound(_)),
       assert(bound(1000)),
       unset(ignorePenalties),
       set(sumScores),
       retractall(tagPenalty(_)),
       set(unknownWords)),
      arabicTest).

GOOD 2018, BAD 5824, NEITHER 11332
Total number of edges used: 19174
```

3. Ignoring hand-coded penalties, no tagging, combining penalties by addition

```
runTest((retractall(soft(_, _)),
       assert(soft(_, 77)),
       retractall(bound(_)),
       assert(bound(1000)),
       set(ignorePenalties),
       set(sumScores),
       retractall(tagPenalty(_)),
       set(unknownWords)),
      arabicTest).
```

```
GOOD 1871, BAD 5466, NEITHER 16814
Total number of edges used: 24151
```

4. Ignoring hand-coded penalties, no tagging, combining penalties by averaging

```
runTest((retractall(soft(_, _)),
       assert(soft(_, 77)),
       retractall(bound(_)),
       assert(bound(1000)),
       set(ignorePenalties),
       unset(sumScores),
       retractall(tagPenalty(_)),
       set(unknownWords)),
      arabicTest).
```

```
GOOD 1871, BAD 5466, NEITHER 16814
Total number of edges used: 24151
```

5. Using Hand-coded penalties, tagging with a score of 40, combining penalties by averaging

```
runTest((retractall(soft(_, _)),
    assert(soft(_, 77)),
    retractall(bound(_)),
    assert(bound(1000)),
    unset(ignorePenalties),
    unset(sumScores),
    retractall(tagPenalty(_)),
            assert(tagPenalty(40)),
    set(unknownWords)),
   arabicTest).
```

```
GOOD 1960, BAD 4969, NEITHER 9198
```

```
Total number of edges used: 16127
```

6. Ignoring Hand-coded penalties, tagging with a score of 40, combining penalties by averaging

```
runTest((retractall(soft(_, _)),
    assert(soft(_, 77)),
    retractall(bound(_)),
    assert(bound(1000)),
    set(ignorePenalties),
    unset(sumScores),
    retractall(tagPenalty(_)),
            assert(tagPenalty(40)),
    set(unknownWords)),
   arabicTest).

GOOD 1840, BAD 4813, NEITHER 13452
Total number of edges used: 20105
```

7. Using Hand-coded penalties, tagging with a score of 40, combining penalties by addition

```
runTest((retractall(soft(_, _)),
    assert(soft(_, 77)),
    retractall(bound(_)),
    assert(bound(1000)),
    unset(ignorePenalties),
    set(sumScores),
    retractall(tagPenalty(_)),
            assert(tagPenalty(40)),
    set(unknownWords)),
   arabicTest).

GOOD 1894, BAD 6819, NEITHER 15385
```

```
Total number of edges used: 24098
```

8. Ignoring Hand-coded penalties, tagging with a score of 40, combining penalties by addition

```
runTest((retractall(soft(_, _)),
    assert(soft(_, 77)),
    retractall(bound(_)),
    assert(bound(1000)),
    set(ignorePenalties),
    set(sumScores),
    retractall(tagPenalty(_)),
            assert(tagPenalty(40)),
    set(unknownWords)),
   arabicTest).

GOOD 1830, BAD 6214, NEITHER 19047
Total number of edges used: 27091
```

### 8.2.1 Discussion of the results

After running the experiments with three different parameters: using or not using tagging, using or ignoring hand-coded penalties, and averaging or adding the penalties when combining edges. We are looking for the number of edges created depending on the different parameters. The results are shown in Table 8.2.

#### 8.2.1.1 The effects of the type of combining edges: Adding penalties vs Averaging penalties

Figure 8.2 clearly shows that using the average of penalties when combining edges is better than adding them. When we used hand coded penalties, and no tagging, averaging penalties resulted in 18649 edges whereas adding them resulted in 19174 edges. When we used tagging and hand coded penalties, adding the penalties gave 24098 edges, whereas averaging penalties resulted in 16127 edges which is about 33% improvement. Even if we do not use hand-coded penalties, and only use the penalties

| Experiment | Hand-coded penalties | Tagging | Average or Sum | Number of edges |
|---|---|---|---|---|
| 1 | Y | N | Average | 18649 |
| 2 | Y | N | Sum | 19174 |
| 3 | N | N | Does not matter | 24151 |
| 4 | Y | Y | Average | 16127 |
| 5 | Y | Y | Sum | 24098 |
| 6 | N | Y | Average | 20105 |
| 7 | N | Y | Sum | 27091 |

Table 8.2: The effects of tagging and different parameters on the number of edges created

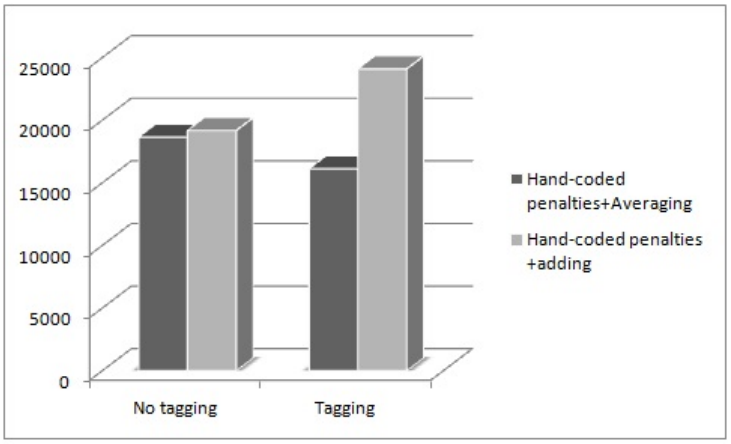



Figure 8.2: Adding penalties vs Averaging penalties

suggested by the tagger, using the average method will improve the result from 27091 edges (when adding penalties) to 20105 edges, which is about 26% improvement.

#### 8.2.1.2 Using hand-coded penalties vs Ignoring hand-coded penalties

Figure 8.3 shows that ignoring the hand-coded penalties makes things worse. Without the use of the tagger, and assuming we use the average method of combining edges, ignoring the hand coded penalties would result in 24151 edges whereas if we use the hand-coded penalties, it would improve to 18649 edges which is nearly 23% improvement. Using the hand-coded penalties will also help improving the result when we use the tagger, which is about 20% improvement.

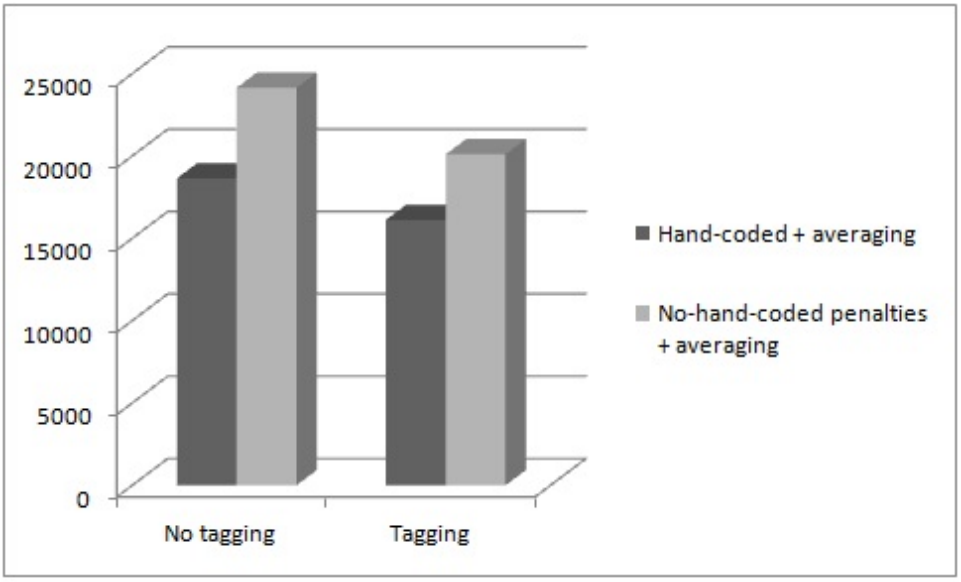



Figure 8.3: Using hand-coded penalties vs Ignoring them

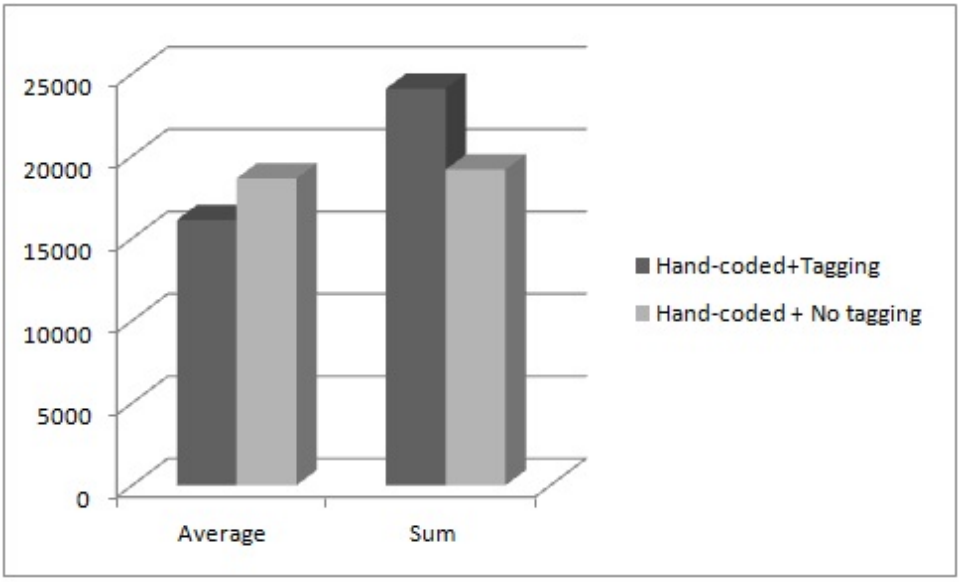



Figure 8.4: The effects of using the tagger

#### 8.2.1.3 The effects of using the tagger

Figure 8.4 shows that using the tagger sometimes makes things worse and other times improves the result. It all depends on the other parameters. It seems that using the tagger improves things when using the hand-coded penalties and combining scores by averaging, but when combining scores by adding it makes things worse: : from 19174 edges when not using the tagger to 24098 edges when using the tagger. However using the average method of combining edges improves the result from 18649 edges to 16127 edges which is about 13.5%. Therefore, we can conclude that using the tagger does help if we use the right parameters: hand-coded penalties, and averaging when combining items.

By going through the analyses to find out why tagging and adding penalties did not improve the results of the parser, It seems that there are several sentences that tagging makes things worse, i.e. the number of edges created increases dramatically. This is because there are loads of instances where the tagger suggested good scores to nouns. This resulted in the parser starting with these edges. Combining the score of two edges is simply adding the score of each one. If the tagger was confident about some edges being nouns, combining them to make other nouns will lead to a huge number of unnecessary edges.

## 8.3 Integrating the edge classification

### 8.3.1 Experiment

The aim of this experiment is to investigate the effects of using machine learning techniques on the speed of the parser. This involves a set of 800 sentences which is divided into a training set of 600 sentences, and a test set of 200 sentences. We then run the first parse of the training set and collect information about the moves. The moves of the parsing are divided into three types: moves that lie on the actual solution are called good moves, moves that lead directly away from it are called bad moves, and moves that follow on from bad moves (neither good nor bad moves) are called neutral. The aim here is to encourage the good moves and discourage the bad moves. One can do this by introducing a notion of penalty or score. Every edge is associated with a score which is represented in a sign. The parser gives preference to lower scores. By analysing the output of the original parser, and why it is slow when given long sentences, it was noted that many edges were created which were not used in the final

analysis. The aim here is to learn facts about edges and use these to guide the parser by giving preference to good edges (by providing it with a negative score) and also dis-preferring bad edges (by penalising it with a positive number) as follows:

```
goodOrBad(Z) :-
    (\+ \+ (call_residue(getBaseARFF(Z, ARFF), _),
       tryj48(ARFF, STATUS)) ->
(useJ48(N) ->
    ((STATUS = good) ->
penalise(Z, N);
(STATUS = bad) ->
penalise(Z, -1*N);
true);
    true);
true).
```

The experiment involves:

- Parsing the training set of sentences and collect information regarding the moves as described above. These are statistical facts about good and bad edges.
- Using decision trees from WEKA to learn classification rules.

As mentioned in Chapter 6 the general syntactic framework lies somewhere between HPSG and pure categorial grammar, but with a rather radical approach to extraposition. An edge is represented as a sign which contains a set of features. Some of these features we are learning about are mentioned below:

```
'{'structure(positions(start(0),
                    end(3),
                    span(7),
                    +compact,
                    xstart(0),
                    xend(3)),

          core('{'structure(positions(start(2), end(3), _C, _D, _E, _F),
                        forms(text('.'),
                              underlying([character(_G,
                                  .
```

```
                                        .
                                        .
                                   surface('.'),
                                   _O),
                              tag(tag(ptag(_P), _Q, _R, _S, ntag([]))),
                                        .
                                        .
                         syntax(nonfoot(head(cat(punct),
                                             _A1,


                          _
 syntax(nonfoot(head(cat(punct),

 meaning(semantics(mood(_F5, _G5, -interrogative, _H5, -irreal, _I5, +main)),
         _L9,
         uses(_M9, _N9, -predicative, -modifier, _O9, _P9, _Q9, _R9, _S9),
         sort(_T9)),
 remarks(_U9,
         _V9,
         failures((_W9 + (_X9 + _Y9))),
         score(([20 | _Z9] + ([20, 4, 4 | _A10] + [20 | _B10]))))'}'
```

### 8.3.2 Parsing using machine learning but not using the tagger

#### 8.3.2.1 The effects on the total number of edges

Our aim here is to reduce the number of bad edges created, hence the total number of edges. Before applying the classification rules, 27861 edges were created. After applying the rules, this number was reduced and the result depends on the value of the score (penalty). We got the optimal result of 25264 edges when the penalty was -45. This gives us 9.32% improvement. Varying the score did not have a radical change as shown in Figure 8.5.

#### 8.3.2.2 The effects on the average time per sentence

Our aim in this experiment is to see whether applying the rules will speed up the parser, and also the relationship between the edges created and the speed of the parser. Without applying the rules, the average time per sentence was 1529.39 ms. The optimal speed

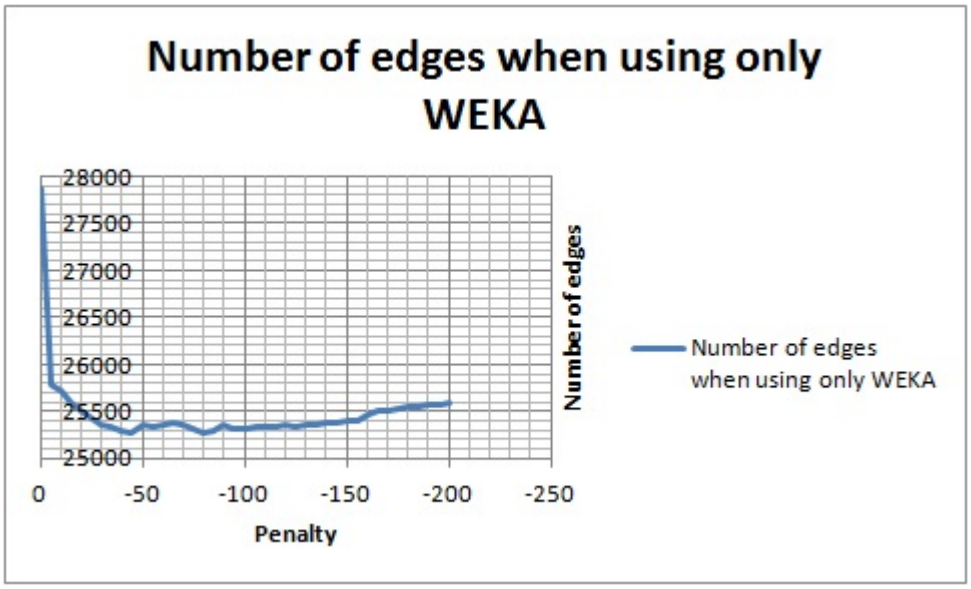



Figure 8.5: The effects of machine learning on the the total number of edges

up was when the penalty was -45 (1036.88 ms) which is approximately a 32% increase in speed. By comparing the two graphs 8.5 and 8.6 we can note here that less edges created the faster the parser is.

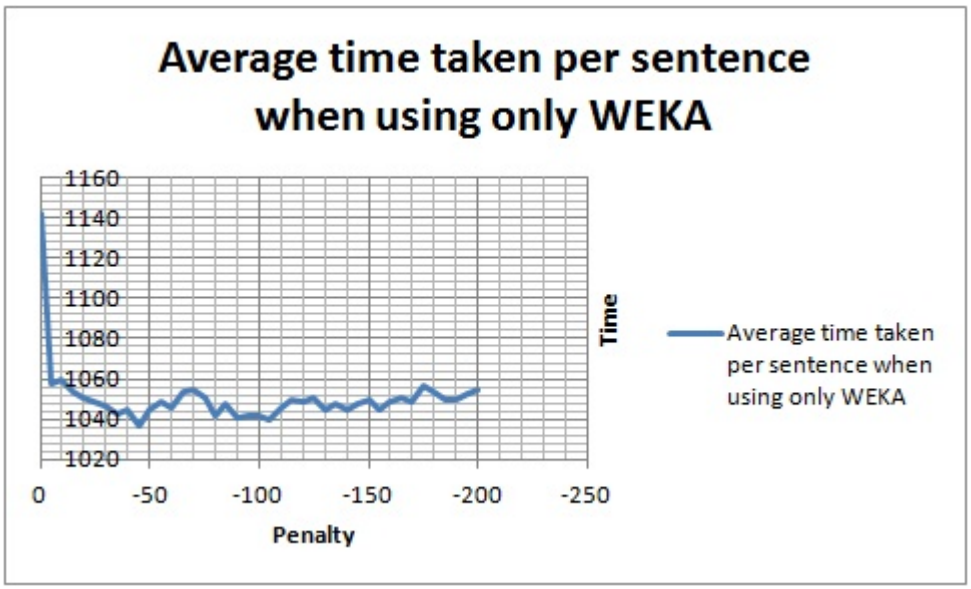



Figure 8.6: The effects of machine learning on the average time per sentence

### 8.3.3 Parsing using machine learning with the tagger

#### 8.3.3.1 The effects on the total number of edges

The third part of these experiments is to investigate the effects of combining both the tagger, and the learning from WEKA. Before applying the classification rules, 27557 edges were created. After applying the rules, we can note that there is some improvement and this depends on the value of the score (penalty). The optimal result is with the penalty -55 where 26504 edges were produced as shown in Figure 8.7. This represents approximately 4% improvement. This is understandable because the tagger already improved the speed, and that is the reason why when applying machine learning the improvement is not as good as when we use machine learning and no tagging.

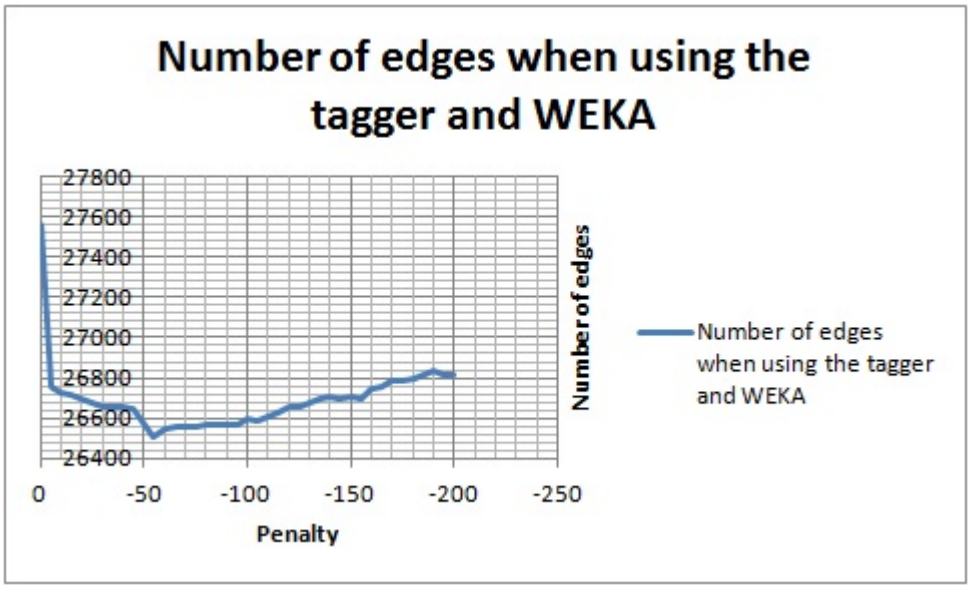



Figure 8.7: Combining both machine learning and the tagger

#### 8.3.3.2 The effects on the average time per sentence

This is to investigate whether that relationship between the number of edges created and the average time per sentence still applies when combining both the tagger and WEKA during parsing. Without applying the machine learning rules, the average time per sentence was 1183.88 ms. The optimal speed up was (1138.14 ms) approximately 4%an increase of speed as shown in Figure 8.8

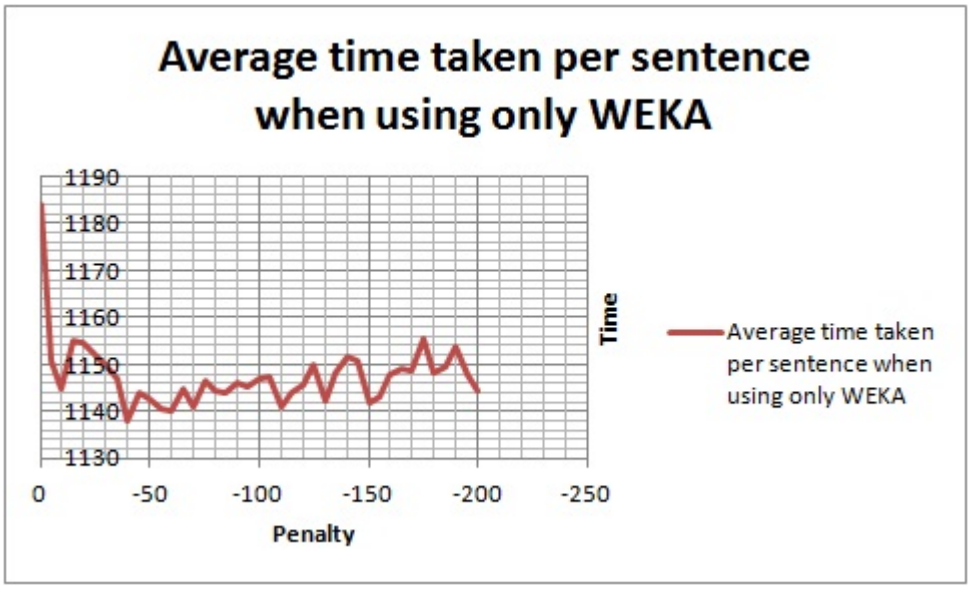



Figure 8.8: Combining both machine learning and the tagger

## 8.4 The effects of experiments on the accuracy

We are trying to improve the speed of a parser. By doing this, we recognise that because we are changing the decisions that this makes, we might lead it to come to a different overall analysis (this can be seen as a decrease in accuracy). As the the aim of this project is not about improving the accuracy, but about the speed of a parser, the last thing you want is considerably speeding up the parser, but the accuracy is affected badly. That is why the aim of this chapter is to investigate the extent of the effects of these experiments on the accuracy of the parser. We are not interested in improving the accuracy, but at least we want to know that if we speed up the parser by these experiments, what is the tradeoff on the accuracy. One way of doing this is to assume that the output of the original parser before applying the experiments is our gold standard.

### 8.4.1 The experiments

The way we are conducting these experiments is that every time we are producing analyses we store the tree information in a conll data format, so that we can compare the tree analysis of the original parser with those of the output after using machine learning. The comparison is between a pair of words. For a given sentence of N words, there are N-1 comparisons, because every word except the head is a daughter

in a tree. For example: out of the 1000 words there are 181 comparisons to be made.

```
total number of comparisons          819
total number of sentences          181
number of right heads              780           0.952380952
number of right labels              772           0.942612943
Segmentation is different             3           0.016574586
```

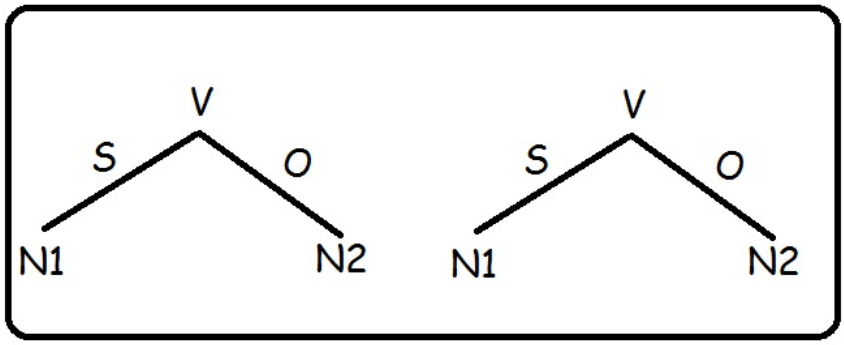



Figure 8.9: Comparing two trees

Figure 8.9 involves comparing the words, their head, and the label. So the questions we are asking here are:

1. How many words have the right head?
2. What is the number of the right labels?
3. How many times a word is present in the gold standard but not in the test set?
4. How many times a word is present in the test set but not present in the gold standard set?
5. How many times a word is segmented differently?

### 8.4.2 Using machine learning and not using the tagger

It is worth noting that improving the speed using machine learning and the tagger has a tradeoff of a slight decrease in the accuracy. This decrease varies depending on the value of the penalty. For instance when the penalty was -5 the number of equal heads

scored 95% accuracy and the number of equal labels with 94%. Figures 8.10 and 8.11 show this variation. There are no words at all that appeared in the gold standard and not in the test set, and vice versa. Different segmentation in the table means that the same surface form may be interpreted in more than one way because of the clitics. For instance the word وجد *wğd* maybe interpreted as وَجَدَ *waǧada* 'found' or the conjunction و *w* 'and' followed by the verb جد *ğd* 'worked hard'.

| Penalty | comparisons | sentences | right heads | head accuracy | right labels | label accuracy | segmentation |
|---|---|---|---|---|---|---|---|
| -5 | 819 | 181 | 780 | 0.952380952 | 772 | 0.942612943 | 3 |
| -10 | 803 | 181 | 754 | 0.938978829 | 743 | 0.925280199 | 6 |
| -15 | 790 | 181 | 734 | 0.929113924 | 723 | 0.915189873 | 8 |
| -20 | 777 | 181 | 717 | 0.922779923 | 706 | 0.908622909 | 9 |
| -30 | 749 | 181 | 692 | 0.923898531 | 683 | 0.91188251 | 13 |
| -35 | 745 | 181 | 688 | 0.923489933 | 679 | 0.911409396 | 14 |
| -40 | 737 | 181 | 679 | 0.921302578 | 670 | 0.909090909 | 15 |
| -45 | 732 | 181 | 672 | 0.918032787 | 663 | 0.905737705 | 16 |
| -50 | 717 | 181 | 657 | 0.916317992 | 648 | 0.90376569 | 17 |
| -55 | 692 | 181 | 631 | 0.911849711 | 622 | 0.898843931 | 22 |
| -60 | 680 | 181 | 616 | 0.905882353 | 607 | 0.892647059 | 25 |
| -65 | 675 | 181 | 611 | 0.905185185 | 602 | 0.891851852 | 26 |
| -70 | 669 | 181 | 602 | 0.899850523 | 593 | 0.886397608 | 27 |
| -75 | 675 | 181 | 608 | 0.900740741 | 599 | 0.887407407 | 26 |
| -80 | 675 | 181 | 602 | 0.891851852 | 593 | 0.878518519 | 26 |
| -85 | 675 | 181 | 601 | 0.89037037 | 592 | 0.877037037 | 26 |
| -90 | 675 | 181 | 601 | 0.89037037 | 592 | 0.877037037 | 26 |
| -95 | 669 | 181 | 595 | 0.889387145 | 586 | 0.87593423 | 28 |
| -100 | 664 | 181 | 589 | 0.887048193 | 580 | 0.873493976 | 29 |

Figure 8.10: The effect on the accuracy when using machine learning and not the tagger

### 8.4.3 Using machine learning and the tagger

Again the improvement of the parser by using the tagger and machine learning has a slight tradeoff which varies depending on the value of the penalty. This variation is shown in Figures 8.12 and 8.13. We can note that the accuracy decreases when the penalty is reduced. From approximately 97% on the number of heads accuracy when the penalty is -5 to 92% when the penalty is -100.

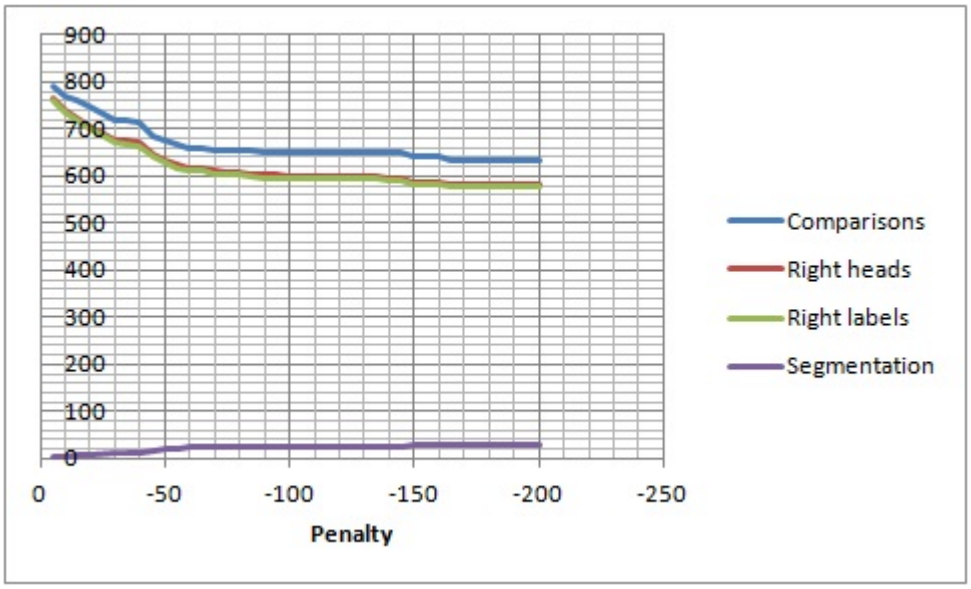



Figure 8.11: Comparing heads, labels, and segmentations when not using the tagger

| Penalty | Sentences | Comparisons | Right heads | Head accuracy | Right labels | Label accuracy | Segmentation |
|---|---|---|---|---|---|---|---|
| -5 | 179 | 789 | 764 | 0.968314322 | 762 | 0.965779468 | 3 |
| -10 | 179 | 771 | 738 | 0.957198444 | 734 | 0.952010376 | 5 |
| -15 | 179 | 759 | 722 | 0.951251647 | 717 | 0.944664032 | 7 |
| -20 | 179 | 749 | 707 | 0.943925234 | 702 | 0.937249666 | 8 |
| -30 | 179 | 718 | 675 | 0.940111421 | 670 | 0.933147632 | 12 |
| -35 | 179 | 718 | 674 | 0.938718663 | 669 | 0.931754875 | 12 |
| -40 | 179 | 714 | 670 | 0.93837535 | 665 | 0.931372549 | 13 |
| -45 | 179 | 686 | 646 | 0.941690962 | 641 | 0.934402332 | 18 |
| -50 | 179 | 675 | 635 | 0.940740741 | 630 | 0.933333333 | 20 |
| -55 | 179 | 667 | 624 | 0.935532234 | 618 | 0.926536732 | 22 |
| -60 | 179 | 658 | 617 | 0.93768997 | 611 | 0.928571429 | 23 |
| -65 | 179 | 659 | 617 | 0.936267071 | 611 | 0.927162367 | 23 |
| -70 | 179 | 653 | 611 | 0.93568147 | 605 | 0.926493109 | 24 |
| -75 | 179 | 654 | 609 | 0.931192661 | 603 | 0.922018349 | 24 |
| -80 | 179 | 654 | 609 | 0.931192661 | 603 | 0.922018349 | 24 |
| -85 | 179 | 654 | 605 | 0.925076453 | 599 | 0.915902141 | 24 |
| -90 | 179 | 651 | 602 | 0.924731183 | 596 | 0.915514593 | 25 |
| -95 | 179 | 651 | 602 | 0.924731183 | 596 | 0.915514593 | 25 |
| -100 | 179 | 651 | 601 | 0.923195084 | 595 | 0.913978495 | 25 |

Figure 8.12: The effect on the accuracy when using machine learning with the tagger

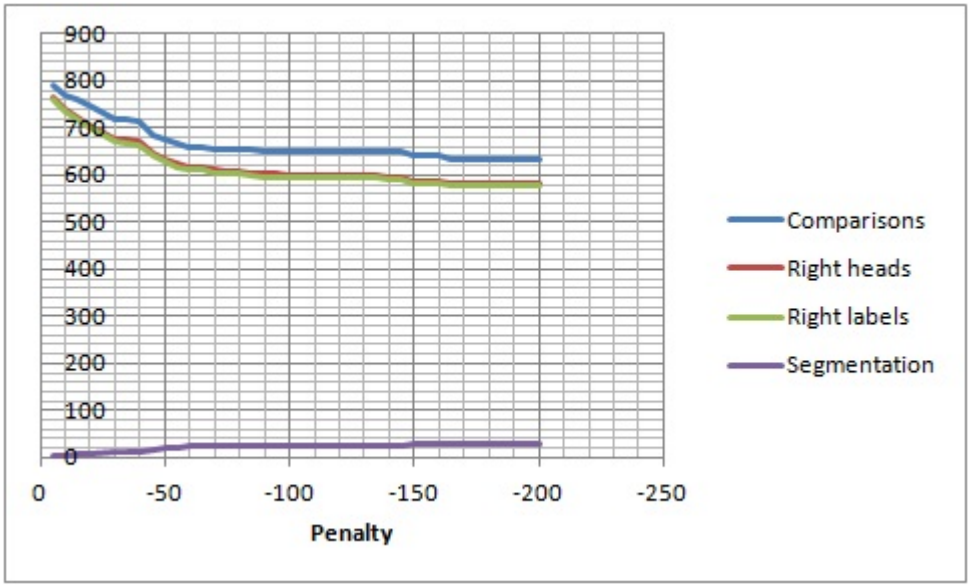



Figure 8.13: Comparing heads, labels and segmentations when using the tagger

# Chapter 9

# Conclusion

My aim throughout this thesis was to investigate ways of optimising a rule based parser for MSA. In order to run such experiments we had to do the following:

- As the previous lexicon only had 295 manually coded entries, and in the absence of MSA lexicons, I extracted a lexicon of 63339 words in a prolog format without any manual intervention. MSA is highly inflectional and derivational, i.e words are produced by following certain patterns and adding the affixes. Therefore creating the lexicon in this way resulted in many weaknesses especially du to large number of weak verbs which they do not follow the same pattern as regular verbs.

- Dealing with the weak verbs by changing or adding some spelling rules. Again, this resulted in conflicting spelling rules and overcoming this one may consider in the future.

- The next step was integrating the output of a tagger with the parser. These initial experiments were awkward to carry out because the original tagger was written in Python, and incorporating it into the Prolog-based parser required us to run it as a server, with the parser as a client. This was inefficient, but more importantly it made the integration very inflexible. Therefore we have converted the application of the tagger in prolog so that it can be integrated with the parser.

- Integrating the classifier with the parser

- Investigating these effects on the accuracy of the parser.

It was noted that adding the tagger to the parser improved the parser's speed. The tagger was tried in two ways:

- Here we trust the tagger, so if the tagger is confident about a tag, we use suggestion directly. In many cases this led to a complete failure where the suggested tag was wrong. For example parsing this sentence:

  كتب الدرس *ktb āldrs* can have the following interpretations:

  1. كَتَبَ الدَّرسَ *kataba ālddarsa* (He wrote the lesson). This is in the active form, and the subject is hidden.
  2. كُتِبَ الدَّرسُ *kutiba ālddarsu* (The lesson was written). This is in the passive form
  3. كَتَّبَ الدَّرسَ *kattaba ālddarsa* (He made someone write the lesson). The verb is active and the subject is hidden.
  4. كُتِّبَ الدَّرسُ *kuttiba ālddarsu* (The lesson was made written) The verb is passive.
  5. كَتَبَ الدَّرسُ *kataba ālddarsu* (The lesson wrote). The parser is not interested in the meaning.
  6. كَتَّبَ الدَّرسُ *kattaba ālddarsu* (The lesson made someone write). Again the parser is not interested in the meaning.
  7. كُتُبُ الدَّرسِ *kutubu ālddarsi* (The books of the lesson). The parser mad a construct NP

  All these are possible outputs of the parser. However, if the tagger is confident enough that كتب *ktb* is noun and الدرس *āldrs* is noun, the only interpretation we get is number 7. In a long sentence assigning all constituents as nouns will lead to a complete failure.

- Using the tagger together with a confusion matrix that indicates the likelihood of a word actually having a tag Ti given that the tagger has suggested Tj.

In the second part of the thesis, we used a novel approach (learning from the solution path: bad and good moves). This resulted in a 9.32% improvement on the total number of edges being produced, and 32% speed up. However by taking the assumption that the original parser is the gold standard, there is a tradeoff in

decreasing the accuracy of the parser. This approach can be useful for under resourced languages.

# Bibliography


[AM12] Ramsay A. Alabbas M. Combining black-box taggers and parsers for modern standard arabic. In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS-2012)*, pages 19–26, 2012.

[Arn] Doug Arnold. *Chart Parsing*. http://webdocs.cs.ualberta.ca/ lindek/650/papers/chartParsing.pdf.

[Att08] Mohammed Attia. Alternate agreement in arabic. Technical report, The University of Manchester, 2008.

[BBG71] G. M. Rubin B. B. Greene. Automatic grammatical tagging of english. Technical report, Providence, Rhode Island, 1971.

[Bee90] Kenneth R Beesley. Finite-state description of arabic morphology. In *Proceedings of the Second Cambridge Conference on Bilingual Computing in Arabic and English*, pages 5–7, 1990.

[Bee98] K.R. Beesley. Arabic morphological analysis on the internet. In *Proceedings of the International Conference on Multi-Lingual Computing (Arabic*, 1998.

[Bee01] Kenneth R. Beesley. Finite-state morphological analysis and generation of arabic at xerox research: status and plans. In *Proceedings of the Workshop on Arabic Natural Language Processing at the 39th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, 2001.

[BhW07] Roy Bar-haim and Yoad Winter. Part-of speech tagging of modern hebrew text. *Journal of Natural Language Engineering*, 2007.

[BM76] L. R. Bahl and R. L. Mercer. Part of speech assignment by a statistical decision algorithm. In *Proceedings IEEE International Symposium on Information Theory*, pages 88–89, 1976.

[Bri92] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155, 1992.

[Bri95] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21:543–565, 1995.

[Buc04] Tim Buckwalter. Buckwalter arabic morphological analyzer version 2.0. Technical Report LDC2004L02, Linguistic Data Consortium, 2004.

[CCA+96] Eugene Charniak, Glenn Carroll, John Adcock, Anthony R. Cassandra, Yoshihiko Gotoh, Jeremy Katz, Michael L. Littman, and John Mccann. Taggers for Parsers. *AI*, 85:45–57, 1996.

[CFL10] Alexander Clark, Chris Fox, and Shalom Lappin. *The Handbook of Computational Linguistics and Natural Language Processing*. Wiley-Blackwell, 2010.

[Cha] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, year = 2000, pages = 132–139.*

[Cha96] Eugene Charniak. Treebank grammars. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036, 1996.

[Cha97] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. pages 598–603. AAAI Press/MIT Press, 1997.

[Chu88] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143, 1988.

[Col96] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 184–191, 1996.

[Col97] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, 1997.

[Col03] Michael Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, pages 589–637, 2003.

[CRHT99] Michael Collins, Lance Ramshaw, Jan Hajic, and Christoph Tillmann. A statistical parser for czech. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 505–512. Association for Computational Linguistics, 1999.

[DeR88] Steven J. DeRose. Grammatical category disambiguation by statistical optimization. *Comput. Linguist.*, 14:31–39, January 1988.

[DF03] Joseph Dichy and Ali Fargaly. Roots & patterns vs. stems plus grammar-lexis specifications: on what basis should a multilingual lexical database centred on arabic be built? In *Proceedings of the MTSummit IX workshop on Machine Translation for Semitic Languages, New-Orleans*, 2003.

[Dia09] M. Diab. Second generation tools (amira 2.0): fast and robust tokenization, pos tagging, and base phrase chunking. In *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools*, pages 285–288, 2009.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, pages 94–102, 1970.

[EBR92] J. Lafferty D. Magerman R. Mercer E. Black, F. Jelinek and S. Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 31—-37, 1992.

[Gar87] Roger Garside. The CLAWS word-tagging system. In *The Computational Analysis of English: a corpus-based approach*, pages 30–41. 1987.

[GKPS85] Gerald Gazdar, Ewan Klein, Geoffry Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, 1985.

[Gui95] Linda Van Guilder. Automated part of speech tagging: A brief overview. Technical report, 1995.

[Hab04] Nizar Habash. Large scale lexeme based arabic morphological generation. In *Proceedings of Traitement Automatique du Langage Naturel TALN04*, 2004.

[Hab10] Nizar Habash. *Introduction to Arabic Natural Language Processing*. Morgan and Claypool Publishers, 2010.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[HNN+07] Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, and et al. Single malt or blended? a study in multilingual parser optimization. In *Proceedings of the Shared Task Session of EMNLP-CoNLL*, pages 933–939, 2007.

[HR06] Nizar Habash and Owen Rambow. Magead: A morphological analyzer and generator for the arabic dialects. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, page 681–688, 2006.

[HR09a] Nizar Habash and Ryan Roth. Catib: The columbia arabic treebank. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 221–224, 2009.

[HR09b] Rambow O. Habash, N. and R. Roth. Mada+tokan: a toolkit for arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. In *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools*, pages 102–109, 2009.

[HSZ+04] Jan Hajič, Otakar Smrž, Peter Zemánek, Jan Šnaidauf, and Emanuel Beška. Prague arabic dependency treebank: Development in data and tools. In *Proceedings of NEMLAR*, pages 110–117, 2004.

[HUK07] Fahim Muhammad Hasan, Naushad UzZaman, and Mumit Khan. Comparison of different pos tagging techniques (n-gram, hmm and brill's tagger) for bangla. In *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, pages 121–126. 2007.

[JK94] Mark Johnson and Martin Kay. Parsing and empty nodes. *Comput. Linguist.*, 20(2):289–300, 1994.

[JLM$^+$] E. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. Decision tree parsing using a hidden derivation model. In *Proceedings of the Workshop on Human Language Technology*, pages 272–277.

[JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 2000.

[Kar90] Fred Karlsson. Constraint grammar as a framework for parsing running text. In *Proceedings of the 13th conference on Computational linguistics - Volume 3*, pages 168–173, 1990.

[KGM06] Seth Kulick, Ryan Gabbard, and Mitch Marcus. Parsing the arabic treebank: Analysis and improvements. In *Proceedings of theTreebanks and Linguistic Theories Conference*, pages 31—-42, 2006.

[Kho01] Shereen Khoja. Apt: Arabic part-of-speech tagger. *Proceedings of the Student Workshop at NAACL*, pages 20–25, 2001.

[KK03] Ronald M. Kaplan and Tracy H. King. Low-level Markup and Large-scale LFG Grammar Processing. In *Proceedings of the LFG 2003 Conference*, pages 238–249, 2003.

[Kos90] Kimmo Koskenniemi. Finite-state parsing and disambiguation. In *Proceedings of the 13th conference on Computational linguistics - Volume 2*, pages 229–232, 1990.

[KR98] Beesley Kenneth R. Arabic morphology using only finite-state operations. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, pages 50–57, 1998.

[LGB94] G. Leech, R. Garside, and M. Bryant. Claws4: The tagging of the british national corpus. In *Proceedings of the 15th Intl. Conference on Computational Linguistics*, pages 622–628, 1994.

[Mag95] David M. Magerman. Statistical decision-tree models for parsing. In *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, 1995.

[Mar87] Ian Marshall. Choice of grammatical word-class without global syntactic analysis: tagging words in the lob corpus. *Comput. Hum*, pages 139–150, 1987.

[MB04] M. Maamouri and A. Bies. Developing an arabic treebank: methods, guidelines, procedures, and tools. In *Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages (Semitic '04)*, pages 2–9, 2004.

[mit]

[Mit77] Tom M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 1*, pages 305–310, 1977.

[Mit82] Tom M. Mitchell. Generalization as search. *Artif. Intell.*, 18(2):203–226, 1982.

[Mit97] T. Mitchell. *Machine Learning (Mcgraw-Hill International Edit)*. McGraw-Hill Education (ISE Editions), 1997.

[MK10] Graff D. Bouziri B. Krouna S. Bies A. Maamouri, M. and S. Kulick. Ldc standard arabic morphological analyzer (sama) version 3.1. Technical Report LDC2010L01, Linguistic Data Consortium, 2010.

[MMS93] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, pages 313–330, 1993.

[NHN+07] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

[Niv03] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149—-160, 2003.

[Niv04] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50—-57, 2004.

[Niv08] Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553, 2008.

[Pea08] John M. Pearce. *Animal Learning and Cognition, 3rd edition: An Introduction*. Psychology Press, 3 edition, 2008.

[PN01] Robbert Prins and Gertjan Van Noord. Unsupervised pos-tagging improves parsing accuracy and parsing efficiency. In *Proceedings of IWPT*, pages 154–165, 2001.

[Pol] Pollard. *Lectures on the Foundations of HPSG*. Ohio State University.

[Pol88] Carl Pollard. *Categorial Grammar and Phrase Structure Grammar: An Excursion on the Syntax-Semantics Frontier*. D. Reidel, 1988.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1992.

[Ram99] Allan Ramsay. Parsing with discontinuous phrases. *Natural Language Engineering*, 5(03):271–300, 1999.

[RS09] A. Ramsay and Y Sabtan. Bootstrapping a lexicon-free tagger for arabic. In *Proceedings of the 9th Conference on Language Engineering*, pages 202—-215, 2009.

[SGM09] Conal Sathi Spence Green and Christopher D. Manning. Np subject detection in verbinitial arabic clauses. In *Proceedings of the Third Workshop on Computational Approaches to Arabic Script-based Languages (CAASL3)*, 2009.

[SKK01] Roger Garside Shereen Khoja and Gerry Knowles. A tagset for the morphosyntactic tagging of arabic. In *Proceedings of Corpus Linguistics 2001*, pages 341–353, 2001.

[SLM82] Derek H. Sleeman, Pat Langley, and Tom M. Mitchell. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, 3(2):48–52, 1982.

[STC65] Walter S. Stolz, Percy H. Tannenbaum, and Frederick V. Carstensen. Stochastic approach to the grammatical coding of english. *Commun. ACM*, pages 399–405, 1965.

[SvdBN07] Abdelhadi Soudi, Antal van den Bosch, and Gunter Neumann. *Arabic Computational Morphology: Knowledge-based and Empirical Methods*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[SW99] Ivan A. Sag and Thomas Wasow. *Syntactic theory: a formal introduction.* CSLI, Stanford, Ca, 1999.

[TLR02] Min Tang, Xiaoqiang Luo, and Salim Roukos. Active learning for statistical natural language parsing. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 120–127, 2002.

[Vou95] Atro Voutilainen. A syntax-based part-of-speech analyser. In *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, EACL '95, 1995.

[Vou98] Atro Voutilainen. Does tagging help parsing?: a case study on finite state parsing. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 25–36. Association for Computational Linguistics, 1998.

[Wau95] Oliver Wauschkuhn. The influence of tagging on the results of partial parsing in german corpora. *4th International Workshop in Parsing Technologies*, 85:45–57, 1995.

[YM03] Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206, 2003.

# Appendix A

# The decision tree

```
J48 pruned tree
------------------

cat2 = noEdge
|   cat1 = interrog: bad (0.0)
|   cat1 = noEdge
|   |   cat0 = interrog: good (8.19/0.08)
|   |   cat0 = xbar(v(+),n(-))
|   |   |   start0 <= 0: good (368.46/127.46)
|   |   |   start0 > 0: bad (1258.98/436.58)
|   |   cat0 = where: good (1.02/0.01)
|   |   cat0 = xbar(v(+),n(+)): bad (705.67/51.36)
|   |   cat0 = xbar(?,n(+)): bad (0.0)
|   |   cat0 = punct: good (405.58/4.2)
|   |   cat0 = yA: bad (3.07/1.04)
|   |   cat0 = interjection(?): bad (133.15/1.77)
|   |   cat0 = xbar(v(?),?): bad (167.97/49.23)
|   |   cat0 = whichphrase: bad (0.0)
|   |   cat0 = xbar(v(-),n(+))
|   |   |   start0 <= 7
|   |   |   |   end0 <= 4: good (1703.44/651.11)
|   |   |   |   end0 > 4
|   |   |   |   |   start0 <= 4: bad (626.0/251.49)
|   |   |   |   |   start0 > 4
|   |   |   |   |   |   end0 <= 7
|   |   |   |   |   |   |   wh0 = -: good (463.16/152.67)
|   |   |   |   |   |   |   wh0 = +: bad (3.0)
|   |   |   |   |   |   end0 > 7
|   |   |   |   |   |   |   displ0 = dir(moved(?),?,?)
|   |   |   |   |   |   |   |   start0 <= 6: bad (68.62/23.6)
|   |   |   |   |   |   |   |   start0 > 6
|   |   |   |   |   |   |   |   |   end0 <= 8: good (184.34/85.54)
|   |   |   |   |   |   |   |   |   end0 > 8: bad (18.34/6.87)
|   |   |   |   |   |   |   displ0 = dir(?,after(-),?): bad (12.52/3.47)
|   |   |   |   |   |   |   displ0 = dir(moved(-),?,?): bad (0.0)
|   |   |   start0 > 7
|   |   |   |   wh0 = -
|   |   |   |   |   start0 <= 9
|   |   |   |   |   |   end0 <= 10: good (249.88/76.0)
|   |   |   |   |   |   end0 > 10
|   |   |   |   |   |   |   start0 <= 8: good (2.0)
|   |   |   |   |   |   |   start0 > 8: bad (16.0/3.0)
|   |   |   |   |   start0 > 9: good (159.6/31.0)
|   |   |   |   wh0 = +: bad (2.0)
|   cat1 = xbar(v(+),n(-))
|   |   offset1 <= -1
|   |   |   start0 <= 2: neither (31.54/4.78)
|   |   |   start0 > 2: good (20.07/7.65)
|   |   offset1 > -1
|   |   |   span0 <= 504
```

```
|   |   |   |   end0 <= 4: good (21.0/5.0)
|   |   |   |   end0 > 4
|   |   |   |   |   start0 <= 2
|   |   |   |   |   |   cat0 = interrog: bad (0.0)
|   |   |   |   |   |   cat0 = xbar(v(+),n(-)): bad (0.0)
|   |   |   |   |   |   cat0 = where: bad (0.0)
|   |   |   |   |   |   cat0 = xbar(v(+),n(+)): bad (0.0)
|   |   |   |   |   |   cat0 = xbar(?,n(+)): bad (0.0)
|   |   |   |   |   |   cat0 = punct: bad (0.0)
|   |   |   |   |   |   cat0 = yA: bad (0.0)
|   |   |   |   |   |   cat0 = interjection(?): bad (0.0)
|   |   |   |   |   |   cat0 = xbar(v(?),?): bad (0.0)
|   |   |   |   |   |   cat0 = whichphrase: good (2.0)
|   |   |   |   |   |   cat0 = xbar(v(-),n(+)): bad (389.0/68.0)
|   |   |   |   |   start0 > 2
|   |   |   |   |   |   end0 <= 7: good (10.0/2.0)
|   |   |   |   |   |   end0 > 7
|   |   |   |   |   |   |   start0 <= 5: bad (65.0/23.0)
|   |   |   |   |   |   |   start0 > 5: good (3.0)
|   |   |   span0 > 504
|   |   |   |   compact0 = -: good (2.0)
|   |   |   |   compact0 = +: bad (980.0/58.0)
|   cat1 = where: bad (0.0)
|   cat1 = xbar(v(+),n(+)): bad (0.0)
|   cat1 = xbar(?,n(+)): bad (0.0)
|   cat1 = yA: bad (0.0)
|   cat1 = xbar(v(?),?): bad (0.0)
|   cat1 = whichphrase: bad (0.0)
|   cat1 = xbar(v(-),n(+))
|   |   offset1 <= -1
|   |   |   start0 <= 1: neither (22.46/2.87)
|   |   |   start0 > 1
|   |   |   |   mod0 = -
|   |   |   |   |   end0 <= 8: good (20.07/2.39)
|   |   |   |   |   end0 > 8
|   |   |   |   |   |   start0 <= 2: neither (2.87)
|   |   |   |   |   |   start0 > 2: good (11.95/2.87)
|   |   |   |   mod0 = +: bad (4.3/0.96)
|   |   offset1 > -1
|   |   |   start0 <= 0
|   |   |   |   end0 <= 4: bad (256.0/31.0)
|   |   |   |   end0 > 4
|   |   |   |   |   end0 <= 5: good (79.0/34.0)
|   |   |   |   |   end0 > 5: bad (64.0/6.0)
|   |   |   start0 > 0: bad (2173.0/86.0)
cat2 = xbar(v(+),n(-))
|   start0 <= 0
|   |   end1 <= 6
|   |   |   start1 <= 1: good (378.37/77.37)
|   |   |   start1 > 1
|   |   |   |   displ1 = dir(moved(+),after(+),before(?)): good (0.0)
|   |   |   |   displ1 = dir(moved(+),after(?),before(+)): good (0.0)
|   |   |   |   displ1 = dir(moved(-),after(-),before(?)): good (24.0/3.0)
|   |   |   |   displ1 = dir(moved(+),after(-),before(+)): good (0.0)
|   |   |   |   displ1 = dir(?,after(-),?): good (0.0)
|   |   |   |   displ1 = dir(moved(?),?,?): good (0.0)
|   |   |   |   displ1 = dir(moved(-),after(?),before(?))
|   |   |   |   |   end0 <= 3: neither (140.0/47.0)
|   |   |   |   |   end0 > 3
|   |   |   |   |   |   span1 <= 32
|   |   |   |   |   |   |   mod0 = -: good (103.0/32.0)
|   |   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   |   start1 <= 2: good (6.0)
|   |   |   |   |   |   |   |   start1 > 2: neither (15.0/6.0)
|   |   |   |   |   |   span1 > 32
|   |   |   |   |   |   |   mod0 = -: neither (34.0/10.0)
|   |   |   |   |   |   |   mod0 = +: good (11.0/4.0)
|   |   end1 > 6: good (194.0/7.0)
|   start0 > 0
|   |   start0 <= 2
|   |   |   offset1 <= -10
|   |   |   |   end0 <= 12: bad (12.0)
```

```
|   |   |   |   end0 > 12
|   |   |   |   |   mod0 = -: good (3.0)
|   |   |   |   |   mod0 = +: neither (12.0/1.0)
|   |   |   offset1 > -10
|   |   |   |   start0 <= 1
|   |   |   |   |   mod0 = -
|   |   |   |   |   |   end2 <= 5
|   |   |   |   |   |   |   end2 <= 2
|   |   |   |   |   |   |   |   cat1 = interrog: neither (0.0)
|   |   |   |   |   |   |   |   cat1 = noEdge: neither (0.0)
|   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(-))
|   |   |   |   |   |   |   |   |   end0 <= 4: good (7.42/1.0)
|   |   |   |   |   |   |   |   |   end0 > 4: neither (123.0/11.0)
|   |   |   |   |   |   |   |   cat1 = where: neither (0.0)
|   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(+)): neither (0.0)
|   |   |   |   |   |   |   |   cat1 = xbar(?,n(+)): neither (0.0)
|   |   |   |   |   |   |   |   cat1 = yA: neither (0.0)
|   |   |   |   |   |   |   |   cat1 = xbar(v(?),?): good (20.0/2.0)
|   |   |   |   |   |   |   |   cat1 = whichphrase: neither (0.0)
|   |   |   |   |   |   |   |   cat1 = xbar(v(-),n(+))
|   |   |   |   |   |   |   |   |   cat0 = interrog: good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = xbar(v(+),n(-)): good (70.0/20.0)
|   |   |   |   |   |   |   |   |   cat0 = where: good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = xbar(v(+),n(+)): good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = xbar(?,n(+)): good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = punct: good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = yA: good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = interjection(?): good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = xbar(v(?),?): good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = whichphrase: good (0.0)
|   |   |   |   |   |   |   |   |   cat0 = xbar(v(-),n(+)): neither (4.63/0.84)
|   |   |   |   |   |   |   |   end2 > 2: neither (307.0/57.0)
|   |   |   |   |   |   |   end2 > 5
|   |   |   |   |   |   |   |   span2 <= 1016
|   |   |   |   |   |   |   |   |   span1 <= 768
|   |   |   |   |   |   |   |   |   |   end0 <= 6: neither (16.0/6.0)
|   |   |   |   |   |   |   |   |   |   end0 > 6
|   |   |   |   |   |   |   |   |   |   |   end1 <= 5: good (12.0)
|   |   |   |   |   |   |   |   |   |   |   end1 > 5
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = interrog: good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = noEdge: good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(-)): neither (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = where: good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(+)): good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = xbar(?,n(+)): good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = yA: good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = xbar(v(?),?): good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = whichphrase: good (0.0)
|   |   |   |   |   |   |   |   |   |   |   |   cat1 = xbar(v(-),n(+)): good (30.0/9.0)
|   |   |   |   |   |   |   |   |   span1 > 768: neither (5.0)
|   |   |   |   |   |   |   |   span2 > 1016: neither (32.0/1.0)
|   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   end0 <= 8
|   |   |   |   |   |   |   |   end2 <= 3: neither (9.4/3.0)
|   |   |   |   |   |   |   |   end2 > 3: good (15.0)
|   |   |   |   |   |   |   end0 > 8
|   |   |   |   |   |   |   |   span1 <= 1984: neither (26.0/2.0)
|   |   |   |   |   |   |   |   span1 > 1984: good (2.0)
|   |   |   |   start0 > 1
|   |   |   |   |   end1 <= 4
|   |   |   |   |   |   zs0 = -
|   |   |   |   |   |   |   span2 <= 48: good (43.03/13.24)
|   |   |   |   |   |   |   span2 > 48: neither (12.35/3.0)
|   |   |   |   |   |   zs0 = +: neither (102.7/12.16)
|   |   |   |   |   end1 > 4: neither (716.88/66.68)
|   |   start0 > 2
|   |   |   compact0 = -
|   |   |   |   xend0 <= 7: good (2.0)
|   |   |   |   xend0 > 7: neither (26.0/2.0)
|   |   |   compact0 = +
|   |   |   |   displ1 = dir(moved(+),after(+),before(?)): good (0.0)
|   |   |   |   displ1 = dir(moved(+),after(?),before(+))
```

```
|   |   |   |   |   start2 <= 7
|   |   |   |   |   |   start0 <= 5
|   |   |   |   |   |   |   span2 <= 128: good (24.0/7.0)
|   |   |   |   |   |   |   span2 > 128
|   |   |   |   |   |   |   |   offset2 <= -3: neither (2.0/1.0)
|   |   |   |   |   |   |   |   offset2 > -3
|   |   |   |   |   |   |   |   |   span0 <= 240: bad (3.0/1.0)
|   |   |   |   |   |   |   |   |   span0 > 240: good (4.0)
|   |   |   |   |   |   start0 > 5: bad (4.0)
|   |   |   |   |   start2 > 7
|   |   |   |   |   |   start2 <= 9
|   |   |   |   |   |   |   end0 <= 9: good (2.0)
|   |   |   |   |   |   |   end0 > 9: neither (26.0/2.0)
|   |   |   |   |   |   start2 > 9: good (5.0/1.0)
|   |   |   |   displ1 = dir(moved(-),after(-),before(?))
|   |   |   |   |   offset1 <= -4: good (2.0)
|   |   |   |   |   offset1 > -4
|   |   |   |   |   |   cat1 = interrog: neither (0.0)
|   |   |   |   |   |   cat1 = noEdge: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(+),n(-)): neither (0.0)
|   |   |   |   |   |   cat1 = where: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(+),n(+)): good (4.0/1.0)
|   |   |   |   |   |   cat1 = xbar(?,n(+)): neither (0.0)
|   |   |   |   |   |   cat1 = yA: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(?),?): neither (0.0)
|   |   |   |   |   |   cat1 = whichphrase: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(-),n(+)): neither (40.0/1.0)
|   |   |   |   displ1 = dir(moved(+),after(-),before(+))
|   |   |   |   |   wh0 = -: neither (27.0/2.0)
|   |   |   |   |   wh0 = +: good (3.0/1.0)
|   |   |   |   displ1 = dir(?,after(-),?): good (0.0)
|   |   |   |   displ1 = dir(moved(?),?,?): good (0.0)
|   |   |   |   displ1 = dir(moved(-),after(?),before(?))
|   |   |   |   |   start0 <= 4
|   |   |   |   |   |   cat1 = interrog: neither (0.0)
|   |   |   |   |   |   cat1 = noEdge: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(+),n(-))
|   |   |   |   |   |   |   end1 <= 6
|   |   |   |   |   |   |   |   start2 <= 5: good (3.56/0.28)
|   |   |   |   |   |   |   |   start2 > 5: neither (9.0/1.0)
|   |   |   |   |   |   |   end1 > 6: good (21.82/0.28)
|   |   |   |   |   |   cat1 = where: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(+),n(+)): neither (0.0)
|   |   |   |   |   |   cat1 = xbar(?,n(+)): neither (0.0)
|   |   |   |   |   |   cat1 = yA: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(?),?)
|   |   |   |   |   |   |   start1 <= 4: good (8.0/1.0)
|   |   |   |   |   |   |   start1 > 4: neither (25.0/11.0)
|   |   |   |   |   |   cat1 = whichphrase: neither (0.0)
|   |   |   |   |   |   cat1 = xbar(v(-),n(+))
|   |   |   |   |   |   |   end0 <= 6
|   |   |   |   |   |   |   |   zs0 = -
|   |   |   |   |   |   |   |   |   end0 <= 5: good (12.72/4.12)
|   |   |   |   |   |   |   |   |   end0 > 5: neither (60.08/32.08)
|   |   |   |   |   |   |   |   zs0 = +
|   |   |   |   |   |   |   |   |   start0 <= 3: neither (9.14/0.14)
|   |   |   |   |   |   |   |   |   start0 > 3: good (3.16/1.0)
|   |   |   |   |   |   |   end0 > 6
|   |   |   |   |   |   |   |   end0 <= 8
|   |   |   |   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   |   |   |   end0 <= 7
|   |   |   |   |   |   |   |   |   |   |   offset1 <= -3: neither (5.0)
|   |   |   |   |   |   |   |   |   |   |   offset1 > -3
|   |   |   |   |   |   |   |   |   |   |   |   offset1 <= -2: good (17.0/6.0)
|   |   |   |   |   |   |   |   |   |   |   |   offset1 > -2
|   |   |   |   |   |   |   |   |   |   |   |   |   start0 <= 3: good (13.0/5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   start0 > 3: neither (16.14/3.14)
|   |   |   |   |   |   |   |   |   |   end0 > 7
|   |   |   |   |   |   |   |   |   |   |   start0 <= 3
|   |   |   |   |   |   |   |   |   |   |   |   start1 <= 6: neither (3.0)
|   |   |   |   |   |   |   |   |   |   |   |   start1 > 6: good (3.0/1.0)
|   |   |   |   |   |   |   |   |   |   start0 > 3: good (7.14)
```

```
|   |   |   |   |   |   |   |   |   mod0 = +: good (2.0)
|   |   |   |   |   |   |   |   end0 > 8: neither (84.7/25.7)
|   |   |   |   |   start0 > 4
|   |   |   |   |   |   start1 <= 9: good (138.46/32.53)
|   |   |   |   |   |   start1 > 9
|   |   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   |   offset1 <= -4: good (3.0)
|   |   |   |   |   |   |   |   offset1 > -4
|   |   |   |   |   |   |   |   |   start0 <= 9: neither (26.98/9.98)
|   |   |   |   |   |   |   |   |   start0 > 9: good (4.42)
|   |   |   |   |   |   |   mod0 = +: good (6.14/1.14)
cat2 = xbar(v(+),n(+)): neither (0.0)
cat2 = xbar(?,n(+)): good (1.01/0.01)
cat2 = yA: neither (6.06/1.03)
cat2 = punct: good (400.23/2.52)
cat2 = xbar(v(-),n(+))
|   start0 <= 0
|   |   span2 <= 6
|   |   |   cat0 = interrog: good (0.0)
|   |   |   cat0 = xbar(v(+),n(-)): neither (7.27)
|   |   |   cat0 = where: good (0.0)
|   |   |   cat0 = xbar(v(+),n(+)): good (0.0)
|   |   |   cat0 = xbar(?,n(+)): good (0.0)
|   |   |   cat0 = punct: good (0.0)
|   |   |   cat0 = yA: good (0.0)
|   |   |   cat0 = interjection(?): good (0.0)
|   |   |   cat0 = xbar(v(?),?): good (0.0)
|   |   |   cat0 = whichphrase: good (0.0)
|   |   |   cat0 = xbar(v(-),n(+))
|   |   |   |   end0 <= 8
|   |   |   |   |   mod0 = -
|   |   |   |   |   |   end0 <= 3: good (197.73/45.73)
|   |   |   |   |   |   end0 > 3
|   |   |   |   |   |   |   span1 <= 12: neither (11.0/1.0)
|   |   |   |   |   |   |   span1 > 12: good (21.73/2.73)
|   |   |   |   |   mod0 = +
|   |   |   |   |   |   end0 <= 2: good (56.0/8.0)
|   |   |   |   |   |   end0 > 2
|   |   |   |   |   |   |   end0 <= 3: neither (49.0/18.0)
|   |   |   |   |   |   |   end0 > 3: good (17.0)
|   |   |   |   end0 > 8
|   |   |   |   |   mod0 = -: neither (13.0/4.0)
|   |   |   |   |   mod0 = +: good (3.0)
|   |   span2 > 6: good (458.0/6.0)
|   start0 > 0
|   |   start2 <= 1
|   |   |   start1 <= 2: good (177.73/77.45)
|   |   |   start1 > 2
|   |   |   |   end0 <= 5
|   |   |   |   |   mod0 = -: neither (4615.0/351.0)
|   |   |   |   |   mod0 = +: good (12.0/4.0)
|   |   |   |   end0 > 5
|   |   |   |   |   start1 <= 5
|   |   |   |   |   |   span1 <= 32
|   |   |   |   |   |   |   mod0 = -: neither (8.0/4.0)
|   |   |   |   |   |   |   mod0 = +: good (6.0/1.0)
|   |   |   |   |   |   span1 > 32: good (15.0/2.0)
|   |   |   |   |   start1 > 5
|   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   start1 <= 7: neither (9.0)
|   |   |   |   |   |   |   start1 > 7: good (11.0)
|   |   |   |   |   |   mod0 = +: neither (34.0/1.0)
|   |   start2 > 1
|   |   |   span2 <= 1984
|   |   |   |   span0 <= 896
|   |   |   |   |   start2 <= 4
|   |   |   |   |   |   offset2 <= -1: good (41.0/6.0)
|   |   |   |   |   |   offset2 > -1
|   |   |   |   |   |   |   offset1 <= -4
|   |   |   |   |   |   |   |   end0 <= 7: neither (7.0/1.0)
|   |   |   |   |   |   |   |   end0 > 7
|   |   |   |   |   |   |   |   |   span0 <= 496: good (54.0/1.0)
```

```
|   |   |   |   |   |   |   |   |   span0 > 496
|   |   |   |   |   |   |   |   |   |   start1 <= 6: neither (4.0)
|   |   |   |   |   |   |   |   |   |   start1 > 6
|   |   |   |   |   |   |   |   |   |   |   offset1 <= -6: neither (4.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   offset1 > -6: good (7.0/2.0)
|   |   |   |   |   |   |   |  offset1 > -4
|   |   |   |   |   |   |   |   |   cat1 = interrog: neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = noEdge: neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(-))
|   |   |   |   |   |   |   |   |   |   start0 <= 2: neither (33.64/3.55)
|   |   |   |   |   |   |   |   |   |   start0 > 2: good (10.77/2.77)
|   |   |   |   |   |   |   |   |   cat1 = where: neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = xbar(v(+),n(+))
|   |   |   |   |   |   |   |   |   |   start0 <= 3: good (21.0)
|   |   |   |   |   |   |   |   |   |   start0 > 3
|   |   |   |   |   |   |   |   |   |   |   span1 <= 192: neither (5.03/1.0)
|   |   |   |   |   |   |   |   |   |   |   span1 > 192: good (4.0)
|   |   |   |   |   |   |   |   |   cat1 = xbar(?,n(+)): neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = yA: neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = xbar(v(?),?): neither (0.0)
|   |   |   |   |   |   |   |   |   cat1 = whichphrase: good (1.0/0.0)
|   |   |   |   |   |   |   |   |   cat1 = xbar(v(-),n(+))
|   |   |   |   |   |   |   |   |   |   span2 <= 16
|   |   |   |   |   |   |   |   |   |   |   start1 <= 3: good (208.18/96.09)
|   |   |   |   |   |   |   |   |   |   |   start1 > 3
|   |   |   |   |   |   |   |   |   |   |   |   end0 <= 8
|   |   |   |   |   |   |   |   |   |   |   |   |   span1 <= 96
|   |   |   |   |   |   |   |   |   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   end0 <= 6
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   span1 <= 32: good (235.73/104.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   span1 > 32: neither (77.36/37.36)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   end0 > 6: good (31.36/5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   offset1 <= -2: good (21.0/9.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   offset1 > -2: neither (116.36/35.36)
|   |   |   |   |   |   |   |   |   |   |   |   |   span1 > 96
|   |   |   |   |   |   |   |   |   |   |   |   |   |   end0 <= 7: neither (59.0/9.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   end0 > 7
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   offset1 <= -2: neither (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   offset1 > -2: good (23.36/7.0)
|   |   |   |   |   |   |   |   |   |   |   |   end0 > 8: neither (46.0/10.0)
|   |   |   |   |   |   |   |   |   |   span2 > 16
|   |   |   |   |   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   |   |   |   |   start1 <= 6
|   |   |   |   |   |   |   |   |   |   |   |   span2 <= 48
|   |   |   |   |   |   |   |   |   |   |   |   |   span1 <= 64: neither (142.94/25.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   span1 > 64: good (10.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   |   span2 > 48: neither (32.0/6.0)
|   |   |   |   |   |   |   |   |   |   |   start1 > 6: good (18.0/4.0)
|   |   |   |   |   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   |   |   |   |   start1 <= 5
|   |   |   |   |   |   |   |   |   |   |   |   end0 <= 6
|   |   |   |   |   |   |   |   |   |   |   |   |   start0 <= 2: bad (6.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   start0 > 2: good (5.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   end0 > 6: bad (39.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   start1 > 5
|   |   |   |   |   |   |   |   |   |   |   |   start1 <= 6: good (12.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   |   start1 > 6: neither (5.0)
|   |   |   |   |   start2 > 4
|   |   |   |   |   |   span2 <= 256
|   |   |   |   |   |   |   mod0 = -: good (209.64/56.18)
|   |   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   |   start0 <= 7: good (55.09/22.73)
|   |   |   |   |   |   |   |   start0 > 7: bad (8.0/2.0)
|   |   |   |   |   |   span2 > 256
|   |   |   |   |   |   |   mod0 = -
|   |   |   |   |   |   |   |   span2 <= 384: neither (21.0/7.0)
|   |   |   |   |   |   |   |   span2 > 384: good (7.0)
|   |   |   |   |   |   |   mod0 = +: bad (2.0)
|   |   |   |   span0 > 896
|   |   |   |   |   start0 <= 5
|   |   |   |   |   |   span1 <= 2032: neither (530.82/34.45)
```

```
|   |   |   |   |   |   span1 > 2032
|   |   |   |   |   |   |   start1 <= 3: neither (18.0)
|   |   |   |   |   |   |   start1 > 3: good (10.18/2.09)
|   |   |   |   |   start0 > 5
|   |   |   |   |   |   end0 <= 10
|   |   |   |   |   |   |   start1 <= 8
|   |   |   |   |   |   |   |   end2 <= 7: good (6.55/1.45)
|   |   |   |   |   |   |   |   end2 > 7: neither (24.0/2.0)
|   |   |   |   |   |   |   start1 > 8: bad (4.0/1.0)
|   |   |   |   |   |   end0 > 10
|   |   |   |   |   |   |   mod0 = -: good (131.64/26.18)
|   |   |   |   |   |   |   mod0 = +
|   |   |   |   |   |   |   |   end0 <= 11
|   |   |   |   |   |   |   |   |   span2 <= 128: bad (5.0/1.0)
|   |   |   |   |   |   |   |   |   span2 > 128: neither (22.0/5.0)
|   |   |   |   |   |   |   |   end0 > 11
|   |   |   |   |   |   |   |   |   end0 <= 12: good (25.0/1.0)
|   |   |   |   |   |   |   |   |   end0 > 12
|   |   |   |   |   |   |   |   |   |   start1 <= 10: bad (2.0)
|   |   |   |   |   |   |   |   |   |   start1 > 10: neither (5.0)
|   |   |   span2 > 1984
|   |   |   |   offset1 <= -8: bad (12.0)
|   |   |   |   offset1 > -8: good (52.09/7.36)

Number of Leaves  :  266

Size of the tree :  431


Time taken to build model: 1.58 seconds
Time taken to test model on training data: 0.36 seconds

=== Error on training data ===

Correctly Classified Instances       18527               82.6066 %
Incorrectly Classified Instances      3901               17.3934 %
Kappa statistic                          0.7391
Mean absolute error                      0.1681
Root mean squared error                  0.289
Relative absolute error                 37.8322 %
Root relative squared error             61.2963 %
Total Number of Instances            22428


=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.939     0.059      0.888     0.939     0.913      0.975    neither
                 0.744     0.127      0.746     0.744     0.745      0.886    good
                 0.795     0.075      0.841     0.795     0.817      0.952    bad
Weighted Avg.    0.826     0.087      0.825     0.826     0.825      0.938


=== Confusion Matrix ===

    a    b    c   <-- classified as
 7022  450    4 |    a = neither
  791 5565 1120 |    b = good
   92 1444 5940 |    c = bad



=== Stratified cross-validation ===

Correctly Classified Instances       18329               81.7237 %
Incorrectly Classified Instances      4099               18.2763 %
Kappa statistic                          0.7259
Mean absolute error                      0.1723
Root mean squared error                  0.2966
Relative absolute error                 38.7765 %
Root relative squared error             62.9097 %
Total Number of Instances            22428
```

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.928     0.062      0.882     0.928     0.904      0.971    neither
                 0.732     0.133      0.733     0.732     0.733      0.874    good
                 0.792     0.079      0.834     0.792     0.812      0.947    bad
Weighted Avg.    0.817     0.091      0.816     0.817     0.816      0.93


=== Confusion Matrix ===

    a    b    c   <-- classified as
 6935  534    7 |    a = neither
  830 5472 1174 |    b = good
   96 1458 5922 |    c = bad
```

# Appendix B

# Part of the ARFF file

```
@RELATION status
@ATTRIBUTE compact0 {"-","+"}
@ATTRIBUTE compact1 {"-","+"}
@ATTRIBUTE compact2 {"-","+"}
@ATTRIBUTE displ0 {"dir(?,-,?)","dir(-,?,?)","dir(?,?,?)"}
@ATTRIBUTE displ1 {"dir(+,-,+)","dir(-,-,?)","dir(+,?,+)","dir(-,?,?)",...
@ATTRIBUTE displ2 {"dir(?,-,?)","dir(?,?,?)"}
@ATTRIBUTE cat0 {"interrog","xbar(-,+)","xbar(+,+)","xbar(?,+)", ...
@ATTRIBUTE cat1 {"interrog","xbar(-,+)","xbar(+,+)","noEdge", ...
@ATTRIBUTE cat2 {"xbar(-,+)","noEdge","xbar(?,+)","xbar(+,-)", ...
@ATTRIBUTE gerund0 {"-"}
@ATTRIBUTE mod0 {"-","+"}
@ATTRIBUTE spec0 {"+"}
@ATTRIBUTE spec1 {"+"}
@ATTRIBUTE spec2 {"+"}
@ATTRIBUTE start0 NUMERIC
@ATTRIBUTE start1 NUMERIC
@ATTRIBUTE start2 NUMERIC
@ATTRIBUTE end0 NUMERIC
@ATTRIBUTE end1 NUMERIC
@ATTRIBUTE end2 NUMERIC
@ATTRIBUTE xend0 NUMERIC
@ATTRIBUTE xend1 NUMERIC
@ATTRIBUTE xend2 NUMERIC
@ATTRIBUTE span0 NUMERIC
@ATTRIBUTE span1 NUMERIC
@ATTRIBUTE span2 NUMERIC
@ATTRIBUTE offset1 NUMERIC
@ATTRIBUTE offset2 NUMERIC
@ATTRIBUTE wh0 {"-","+"}
@ATTRIBUTE wh1 {"-","+"}
@ATTRIBUTE wh2 {"-","+"}
@ATTRIBUTE zs0 {"-","+"}
@ATTRIBUTE status {"neither","good","bad"}
@DATA
?,?,?,?,?,?,"xbar(-,+)","noEdge","noEdge","-","-","+",?,?,6,?,?,7,..., "good",
"+",?,?,?,?,?,"punct","noEdge","noEdge","-","-",?,?,?,13,?,?,14,?, ...,"good",
"+",?,?,?,?,?,"xbar(+,-)","noEdge","noEdge","-","-",?,?,?,9,?,?,11, ...,"good",

                    .
                    .
                    .
"+","+",?,?,?,?,"xbar(+,-)","xbar(-,+)","noEdge","-","-","+","+",?,3,3, ...,"bad",
"+","+",?,?,?,?,"xbar(+,-)","xbar(-,+)","noEdge","-","-","+","+",?,3,3,?, ...,"bad",
"+","+",?,?,?,?,"xbar(+,-)","xbar(-,+)","noEdge","-","-","+","+",?,3,3, ...,"bad",
"+","+",?,?,?,?,"xbar(+,-)","xbar(-,+)","noEdge","-","-","+","+",?,2,2,?, ...,"bad",

                    .
                    .
```

.

```
"+",?,"+",?,"dir(-,?,?)",?,"xbar(-,+)","xbar(-,+)","xbar(-,+)","-","-","+", ..., "neither",
"+",?,"+",?,"dir(-,?,?)",?,"xbar(-,+)","xbar(-,+)","xbar(-,+)","-","-","+","+", ...,"neither",
"+",?,"+",?,"dir(-,?,?)",?,"xbar(-,+)","xbar(-,+)","xbar(-,+)","-","-","+","+", ...,"neither",
```

# Appendix C

# A sample of the sentences

```
tests(arabicWords1,
 arabic,
'qAl Alrjl AlEjwz.'>0,
'lqd OSbHtu EAjzA lA Oqdr ElY HrAvp AlOrD.'>0,
'w lA ElY Hml AlOvqAl.'>0,
'hl ObqY rhn Albyt.'>0,
'Om Om$y hA}mA ElY wjhy HtY OsqT mn AltEb.'>0,
'fkr kvyrA.'>0,
't*kara OyAm Tfwlth.'>0,
'kyf kAn yqfz kAlOrnb mn mkAn IlY Axr.'>0,
'wySEd kAlqTT IlY OElY AlO$jAr.'>0,
't*kra $bAbh Hyn kAn ydxl fy ErAk mE zmlA}h.'>0,
'wkyf kAn yfwz Elyhm.'>0,
'wlknh lm ykn yxASm OHdA wlA yEtdy ElY OHd.'>0,
'$rd TwylA mE *kryAth.'>0,
'kAnt HyAtuh xSbp mly}p bAlEml wAln$AT.'>0,
'whA hw AlAn $yx kbyr.'>0,
'lA tsAEdh SHth ElY AlqyAm bOEmAl tHtAj jhdA kbyrA kAlzrAEp  Alty zAwlhA TwylA.'>0,
'kAn OHfAduh ylEbwn Hwlh bSxb.'>0,
'fjOp xTrt lh fkrp jElth ybtsm wystyqZ mn $rwdh.'>0,
'qAl.'>0,
'knt OzrE AlqmH wAl*rp.'>0,
'wjnyt Alkvyr mn *lk wAlywm sOkwn mzArEA jdydA.'>0,
'wsOb*r.'>0,
'bdl AlqmH wAl*rp.'>0,
'AlklmAt AlTybp.'>0,
'nAdY AlOwlAd.'>0,
'xAld.'>0,
'mHmd.'>0,
'fATmp.'>0,
'wA}l.'>0,
'tEAlwA yA ObnA}y.'>0,
'AjtmE AlOTfAl qrb Aljd wtsA&l yrtsm ElY wjh kl mnhm.'>0,
'Abtsm.'>0,
'AbtsmwA.'>0,
'qAl.'>0,
'Endy Sndwq mn AlHkAyAt.'>0,
'rkD wA}l nHw Sndwq Aljd Almwjwd fy Sdr Albyt wHAwl ftHh.'>  0,
'DHk Aljd wqAl.'>0,
'I*A ftHt AlSndwq thrb AlHkAyAt.'>0,
'tEAl.'>0,
'tEAl yA wA}l mEy HkAyp xbOthA fy Sdry sOHkyhA lkm.'>0,
'vm bdO.'>0,
'kAn yAmAkAn fy qdym AlzmAn.'>0,
'tjmE AlOwlAd Okvr Hwl Aljd ystmEwn kl klmp yqwlhA.'>0,
'AlO$jAr tnhD mn jdyd wqf Obw HmdAn ElY $AT} AlbHr.'>0,
'yHdq fy AlOmwAj AlSAxbp why tSfE wjh AlSxwr bqswp.'>0,
'wtsA@l fy nfsh.'>0,
'lmA*A ygDb AlbHr.'>0,
```

```
’hl ygDb mn AlmHtlyn Al*yn dnswA wjhh bbwArjhm AlHrbyp.’>0,
’Om Onhu gADb mnA.’>0,
’km lEbnA mEh.’>0,
’wArtmynA fy HDnhi AlrA}E.’>0,
’wkm gnynA mEh AlOgAny AlsEydp.’>0,
’Astrsl Obw HmdAn fy xwATrh.’>0,
’kAn wjhh AlHzyn jAfA.’>0,
’brzt fyh lHyth AlnAmyp kAlE$b AlyAbs.’>0,
’wkAnt AlbsAtyn mn xlfh msAHp mn AlswAd AlqAtm.’>0,
’lqd OHrqhA AlShAynpu bHjp AxtbA@ AlfdA}yyn AlErb fyhA.’>0,
’wwjwd mstwdEAt *xA}rhm AlHrbyp.’>0,
’lm ykn yHznh AHtrAq bstAnh Al*y Onfq Emrh fy AlEnAyp bh  wHsb.’>0,
’lknh t*kr OwlAdh.’>0,
’km wEdhm bAlmlAbs Aljmylp wAlOlEAb wAlHlwY wAlqSS  AlmSwrp.’>0,
’bEd byE AlmHSwl AlqAdm.’>0,
’mA*A ystTyE On yEml.’>0,
’AlO$jAr tqf mHrwqp mtfHmp.’>0,
’AlfwAkh Outlft.’>0,
’AlOwlAd hAjrwA En Alqryp.’>0,
’wAlShAynp yEyvwn fsAdA fy kl mkAn.’>0,
’hAhw yqf wHydA jAnb Al$AT}.’>0,
’lA ydry mA AlEml.’>0,
’jA@t mwjp qwyp lTmt AlSxwr bEnf.’>0,
’AmtlO wjh Oby HmdAn bAlr*A* AlbArd.’>0,
’msH wjhh.’>0,
’lAmst OSAbEh Almbllp $ftyh Alm$qqtyn.’>0,
’$Er bmlwHp qArSp.’>0,
’kz ElY $ftyh.’>0,
’vm Altft bEynyh AlgA}mtyn Swb AlbsAtyn.’>0,
’frk Eynyh kOnh yxfy dmwEh.’>0,
’twjh IlY bstAnh.’>0,
’wqAl bSwt msmwE.’>0,
’AlydAn AlltAn tHsnAn AlzrAEp tHsnAn HmAyp mA tzrEAnh.’>0,
’Aqtrb mn AlO$jAr AlmHrwqp wAlOsY yEtSr qlbh.’>0,
’kAnt AlO$jAr tbdw qAmAt dAknp mgrwsp fy h*h AlOrD.’>0,
’nql bSrh mn $jrp IlY OxrY.’>0,
’mvl mn ybHv En $y@ mhm fqdh.’>0,
’fjOp Artsmt AbtsAmp EryDp ElY $ftyh.’>0,
’mA*A Hdv.’>0,
’lqd kAnt bDE $jyrAt Sgyrp gDp trfE r&wshA End OqdAm h*h AlO $jAr AlswdA@ AlmtfHmp.’>0,
’lm ydri mA*A yfEl.’>0,
’kAn mvl mn msahu sHr.’>0,
’dh$.’>0,
’rkE jAnb $jyrp.’>0,
’HDnhA brfq.’>0,
’wqblhA kOb yqbilu Abnh AlSgyr bEd gyAb Twyl.’>0,
’AltmEt EynAh knSl skyn.’>0,
’whw yqwl.’>0,
’AlO$jAr tnhD mn jdyd.’>0,
’wEly On OnhD OyDA.’>0,
’vm twjh nHw Alqryp ybHv En rjAl AlmqAwmp.’>0,
’Hyrp AlOzhAr OHsato EulA bAlDiyq.’>0,
’fAlAntZAr SaEob wqAs.’>0,
’kAnt tnZr mn zjAj AlnAf*p IlY bAb AlHdyqp.’>0,
’lElh yufotaHu wtrY ObAhA qd EAd mn Alsafr.’>0,
’kAnt kl dqyqp tmDy bbT@ bAlg.’>0,
’qAlt lhA OmuhA.’>0,
’mAbk.’>0,
’tbdyn mnzEjp.’>0,
’Ohk*A tstqblyn wAldk.’>0,
’ArtAHt lklAm OmhA.’>0,
’vm nZrto HwlhA fy OrjA@ Algrfp.’>0,
’xTrt lhA fkrp.’>0,
’jrt msrEp ltnfy*hA.’>0,
’qAlt fy nfshA.’>0,
’sOjmE TAqp mn OzhAr Almrj Aljmylp.’>0,
’ODEhA fy InA@ ElY TAwlp wAldy.’>0,
’xrjt EulA bEd Ono Oxbrto OmhA.’>0,
’kAnt AlOzhAr AlbyDA@ tbdw mvl bHyrp mn Alvlj AlnASE.’>0,
’xfq qlbu EulA bfrH why tqtrb mn Almrj.’>0,
’$dhA h*A AlmnZr AlrA}E.’>0,
’SArt AlOzhAr wADHp OmAm EulA kAnt ttlOlO kEqd  mn All&l&.’>0,
```

```
'xTrt OfkAr kvyrp fy *hn EulA.'>0,
'sOjElhA tZhr mvl krp jmylp mn AlOzhAr.'>0,
'syfrH bhA Oby kvyrA.'>0,
'sOSnE mnhA EqdA.'>0,
'EqdA mn Alzhr wOTwq bhA Enq Oby.'>0,
'wqft EulA bjAnb OzhAr Almrj.'>0,
'kAnt OzhAr kvyrp mtfrqp tHyT mkAn wqwfhA.'>0,
'jlst ltqTf bEDhA.'>0,
'wHyn mdt ydhA ltqTf Owl zhrp rOt zhrp OxrY Okbr wOjml.'>0,
'qAlt.'>0,
'tlk Alzhrp Ojml mn h*h.'>0,
'vm nhDt wdnt mn Alzhrp AlvAnyp.'>0,
'lknhA lm tqTfhA lOnhA $Ahdt Ojml mnhA OyDA.'>0,
'whk*A kAnt tmDy mn zhrp IlY OxrY.'>0,
'tEbt  wlm tqTfo Oy zhrp.'>0,
'hAhy End AlTrf Algrby mn bHyrp AlOzhAr.'>0,
'Olqto nZrp Twylp ElyhA.'>0,
'kAnt EynAhA tlmEAn bfrH wADH.'>0,
'rOt AlOzhAr Ojml mn qbl.'>0,
'naqlato EynyhA fymA HwlhA.'>0,
'wEAdt tHdq fy bHyrp AlOzhAr.'>0,
'klmt nfshA bSwt wADH.'>0,
'h*h AlOzhAr mvl AlOsmAk stmwt I*A xrjt mino mrjhA.'>0,
'EAdt EulA dwn On tqTf Oyp zhrp.'>0,
'wbynmA kAnt tm$y fy Tryq AlEwdp.'>0,
'Odh$thA r&yp AlOzhAr AlbyDA@ Aljmylp ElY Twl AlTryq.'>0,
'rOatohA EulA wkOnhA tm$y xlfhA.'>0,
'kAnt msrwrp jdA wtsA@lt.'>0,
'hl tm$y AlOzhAr HqA.'>0,
'$y@ wAHd kAn ydwr fy *hnhA.'>0,
'On tdEw ObAhA lzyArp bHyrp AlOzhAr.'>0,
'sOl wlyd Omh.'>0,
'lmA*A Oglq AlSahAynp mdrstnA.'>0,
'lmA*A qtlwA Abn xAlty HsAm.'>0,
'lmA*A fq&wA Eyn rfyqy xAld.'>0,
'kAn yryd On ysOl wysOl wysOl.'>0,
'lkn Omh DAqt *rEA mino Os}lth fqAlt.'>0,
'Owh.'>0,
'Ink tkvr mn AlOs}lp yA wlyd.'>0,
'qAl wlyd.'>0,
'Oryd On OErf kl $y@.'>0,
'kl $y@.'>0,
'yA Omy.'>0,
'wkAnt Om wlyd trgb On tjyb ElY kl Os}lp AbnhA.'>0,
'gyr OnhA  lm tfEl.'>0,
'bl HAwlt On tbEdh En h*h Alm$Akl Almxyfp.'>0,
'Alty tHdv kl ywm.'>0,
'wbdOt tHky lh HkAyp Hwryp AlbHr Alty OETt AlSayAd Alfqyr  knzA ElY On yEydhA IlY wTnhA AlbHr.'>0,
'wmA In Onht AlOm AlHkAyp HtY kAn wlyd qd nAm Hyv bdOt HkAyp  OxrY.'>0,
'rOY wlyd fy Hlmh Hwryp jmylp kHwryp AlHkAyp.'>0,
'qAlt lh AlHwryp.'>0,
'OhlA bk OyhA AlTfl AllTyf.'>0,
'hl j}t tbHv En Alknz.'>0,
'ATlb mA t$A@ lOHqqh lk.'>0,
'SAH wlyd frHA.'>0,
'hl tHqqyn ly mA OTlb fElA.'>0,
'qAlt AlHwryp.'>0,
'nEm wfwrA.'>0,
'qAl wlyd.'>0,
'Oryd On tftHy ObwAb mdrstnA.'>0,
'lnEwd IlyhA.'>0,
'nqrO wnlEb.'>0,
'Oryd On yEwd HsAm IlY AlHyAp.'>0,
'wOn tEwd Eyn rfyqy xAld slymp kmA kAnt.'>0,
'fjOp rOY wlyd Onh ydxl IlY Almdrsp mE zmlA}h vm rOY Abn  xAlth HsAm ynAdyh lylEbA mEA.'>0,
'w$Ahd xAldA ynDm IlyhmA.'>0,
'wkAn slymA mEAfY mn kl sw@.'>0,
'lknh Hyn AstyqZ Erf Onh kAn yHlm.'>0,
'wtmnY lw kAn AlHlm Hqyqp.'>0,
'mAzAl wlyd ysOl lmA*A.'>0,
'sykbr ywmA wyErf kl $y@.'>0,
'Allyl wAlOTfAl mn qdym AlzmAn kAn Allyl HzynA jdA.'>0,
```

```
'kAn ysmE OTfAlA yqwlwn.'>0,
'lA nHb Allyl.'>0,
'wAxrwn yqwlwn.'>0,
'Allyl mwH$ wmxyf.'>0,
'wAlAbA@ wAlOmhAt yHAwlwn IbEAd Alxwf.'>0,
'dwn jdwY wyTlbwna mn AlOwlAd Al*hAb IlY Alnwm.'>0,
'fAllyl mxSS llrAHp wAlnhAr llEml.'>0,
'wtuTfO AlODwA@.'>0,
'fyZhr Allyl xlf AlnwAf* qAtmA ygTy kl $y@.'>0,
'AlO$jAr wAlbywt.'>0,
'wAl$wArE.'>0,
'fyjzE AlOwlAd wyOwwn IlY AlfrA$ mkrhyn.'>0,
'wdA}mA yqwlwn.'>0,
'Allyl mxyf.'>0,
'nHn lA nHb Allyl.'>0,
'tjwala Allylu kvyrA.'>0,
'HkY qSth lkl mn SAdfh.'>0,
'qAl lh Alqmr.'>0,
'lA tHzn yA Sdyqy.'>0,
'sOsAEdk.'>0,
'wsyHbk AlOTfAl.'>0,
'frH Allyl Hyn smE *lk.'>0,
'wbEd mdp OTl Alqmr wsTE bbhA@.'>0,
'smE Allylu AlOTfAla yqwlwn.'>0,
'mA Ojml Allyl fy Dw@ Alqmr.'>0,
'wlkn Alqmr lA ystTyE On ybqY TwylA.'>0,
'wEndmA ynthy mn Emlh kAn y*hb IlY mkAn Axr lybdO EmlA  jdydA.'>0,
'fy$Er Allylu On AlOTfAl EAwdhm Alxwf.'>0,
'wqbl Ono ybHv En Hl kAnt Alnjwm tlmE fy bHr AlsmA@.'>0,
'wAlDfAdE tnqu mgnyp Ojml AlOgAny.'>0,
'wkAn ysmE Swt Albwmp why ttmtm.'>0,
'Allyl jmyl wrA}E.'>0,
'wOntm OyhA AlOTfAl jmylwn fA*hbwA IlY AlfrA$.'>0,
'SAr AlOwlAd yntZrwn Alqmr.'>0,
'wbEDhm yntZr Alnjwm fybdO yEdhA mn nAf*th HtY ygfw.'>0,
'wAxrwn kAnwA ysEdwn bOgAny AlDfAdE wHkmp Albwmp.'>0,
'wEndmA y*hbwn IlY AlfrA$ ybd&wn rHlp AlOHlAm.'>0,
'lw knt HSAnA EndmA dxl sEyd IlY Albyt.'>0,
'kAn mlTx AlvyAb bAlwHl.'>0,
'wmlwv Alwjh OyDA.'>0,
'nZrt Omh Ilyh nZrp xASp.'>0,
'fwqf mrtbkA.'>0,
'qAlt AlOm.'>0,
'mA*A fElt bnfsk.'>0,
'hyA IlY AlHmAm.'>0,
'kAn sEyd ykrh AlAstHmAm kvyrA.'>0,
'wgAlbA mAkAn yhrb IlY AllEb.'>0,
'EndmA y$Er On mwEd AlAstHmAm qd HAn.'>0,
'fhw lA yTyq AlSAbwn.'>0,
'lOnh yxr$ Eynyh.'>0,
'wyqrsh bqswp.'>0,
'wkAnt Om sEyd tSbr Elyh whw ynT wySrx.'>0,
'lA Oryd On OstHm.'>0,
'lA Oryd lA Oryd.'>0,
'wAlAn Erf Onh lA xlAS mn AlAstHmAm.'>0,
'bEd On lwv wjhh wydyh wmlAbsh bAlwHl.'>0,
'dxl IlY AlHmAm wrAH yHdv nfsh.'>0,
'lw knt HSAnA SgyrA OnT wOlEb Hyv O$A@.'>0,
'OnAm fwq AlwHl AlTry.'>0,
'Ojry bsrEp kbyrp.'>0,
'OqDm AlE$b AlgD.'>0,
'lA tjbrny Omy ElY AlAstHmAm.'>0,
'flA ydxl AlSAbwn fy Eyny.'>0,
'lkn lA.'>0,
'lA lA Oryd On Okwn HSAnA.'>0,
'fAlHSAn AlSgyr sykbr.'>0,
'wsyjr Erbp.'>0,
'wyHml AlOvqAl.'>0,
'lqd rOyt HSAnA yjr Erbp AlmAzwt.'>0,
'wAlrjl yDrbh bAlswT bqswp.'>0,
'OnA lA OHb On yDrbny OHd.'>0,
'lw knt klbA SgyrA.'>0,
```

```
'lA.'>0,
'lA.'>0,
'lA Oryd On Okwn klbA.'>0,
'bED AlOwlAd yE*bwn AlklAb AlSgyrp.'>0,
'y$dwnhA mn A*AnhA.'>0,
'wyjrwnhA mn O*nAbhA.'>0,
'lqd $Ahdt klbA jA}EA yOkl mn AlfDlAt Almrmyp fy mjmE  AlqmAmp.'>0,
'Oryd On Okwn nmrA qwyA lA OxAf mn $y@.'>0,
'lA.'>0,
'lA.'>0,
'lA Oryd.'>0,
'rOyt nmrA mHbwsA fy qfS fy Hdyqp AlHywAn.'>0,
'qAl ly Oby.'>0,
'lqd ASTAdh rjl qwy wwDEh fy h*h AlHdyqp.'>0,
'ymkn On yTlq Ely OHd AlSyAdyn AlnAr fOmwt.'>0,
'lA Oryd On Omwt lA Oryd.'>0,
'dxlt AlOm wsEyd mAyzAl wAqfA yHdv nfsh.'>0,
'wkAnt qd smEt kl mAnTq bh mn* AlbdAyp.'>0,
'qAlt.'>0,
'mAbk Oalamo txlE mlAbsk bEd yA HSAny AlSgyr.'>0,
'HAlA.'>0,
'HAlA yA mAmA.'>0,
'lkny OxAf AlSAbwn.'>0,
'Inh ykwy Eyny.'>0,
'qAlt AlOm m$jEp.'>0,
'lA txf.'>0,
'hyA OgmD Eynyk wtSwr nfsk HSAnA SgyrA lTyfA.'>0,
'Ow jrwA mh*bA.'>0,
'lkn IyAk On ttSwr nfsk nmrA *A mxAlb Twylp wHAdp txb}  AlOwsAx tHthA.'>0,
'wtxyf rfAqk bhA.'>0,
'fynfDwn Enk.'>0,
'xlE sEyd mlAbsh.'>0,
'wOgmD Eynyh bsrEp.'>0,
'rOY nfsh HSAnA SgyrA yjry bsrEp.'>0,
'vm jrwA ylHs blsAnh yd Omh.'>0,
'bynmA kAnt AlOm qd gmrt jsdh AlTry brgwp AlSAbwn kAn sEyd  yrgb On yrY nfsh nmrA.'>0,
'wHyn hama btqlyd Swt Alnmr.'>0,
'ftH fmh wEynyh.'>0,
'w$d OSAbE ydyh.'>0,
'Srx bSwt qwy mn l*E AlSAbwn.'>0,
'wOTbq Eynyh bqwp.'>0,
'DHkt AlOm wqd qdrt mAxTr lsEyd.'>0,
'fqAlt bEd IzAlp AlSAbwn bAlmA@ AlfAtr.'>0,
'hl rOyt nfsk nmrA.'>0,
'Smt wlm yjb.'>0,
'vm ftH Eynyh.'>0,
'frkhmA jydA.'>0,
'kAn AlmA@ mnE$A.'>0,
'sr sEyd wOx* ylEb bAlmA@ wtmnY On yxrj IlY AlsAHp lylEb mE  rfAqh.'>0,
'wlm yrgb bEd *lk On ykwn gyr sEyd AlInsAn.'>0,
'wtElm kyf lA yxAf mn AlSAbwn.'>0,
'AlbTp *At AlEjlAt.'>0,
'Alklb *A Al$Er AlTwyl wAldb SAHb mETf Alfrw.'>0,
'wAlsayArp AlHmrA@ wAlbyAnw AlSgyr.'>0,
'qAl n$wAn lOlEAbh.'>0,
'AlAn.'>0,
'nHn OSdqA@.'>0,
'sOElmkm AlrqSa.'>0,
'vm nHtfl bSdAqtnA.'>0,
'qAlt AlbTp *At AlEjlAt.'>0,
'OnA bTp lA OErf gyr AlsbAHp.'>0,
'wlA OHbu gyrhA.'>0,
'qAl n$wAn.'>0,
'wh*h AlEjlAt.'>0,
'mA*A tEmlyn bhA.'>0,
'qAlt AlbTp.'>0,
'OsAbq bhA rfyqAty.'>0,
'SAH Alklb *w Al$Er AlqSyr.'>0,
'wOnA Ojls hnA.'>0,
'OHrs OSdqA}y.'>0,
'wlA Otqn gyr *lk.'>0,
'hz Aldb mETafhu Alvqyl qA}lA.'>0,
```

```
’wOnA lA Otrk mETfy Alvqyl.’>0,
’OxAf Albrd kvyrA.’>0,
’rbmA OSAb bAlzkAm.’>0,
’OTlqt AlsyArp AlHmrA@ SwtA TwylA mn mzmArhA.’>0,
’wOnA jAhzp lITfA@ AlHrA}q.’>0,
’OmA AlbyAnw AlSgyr fqd Zl SAmtA.’>0,
’qAl n$wAn.’>0,
’wOnt yA SAHb AlSwt  Aljmyl.’>0,
’mA*A tqwl.’>0,
’wlm yql AlbyAnw AlSgyr $y}A.’>0,
’dh$ n$wAn mn Smt AlbyAnw.’>0,
’lknh srEAn mA lAHZ mTrqtyn Sgyrtyn jAnb AlbyAnw.’>0,
’Ox*hmA n$wAn wTrq bhmA TrqA xfyfA fwq SfA}H AlbyAnw AlSgyr  fAnbvqt OngAm E*bp.’>0,
’rqS Aldb wAlklb wrqSt AlbTp.’>0,
’lkn AlsyArp rAqbt sEAdp OSdqA}hA bsrwr.’>0,
’dwn On tTlq Swt mzmArhA wbqy n$wAn yEzf OlHAnA jmylp tbEv  fy Alnfs AlfrH.’>0,
’w*At lylp lm tstTE On tnAm.’>0,
’wbqyt jAlsp fy sryrhA.’>0,
’kAn OxwthA ynAmwn IlY jAnbhA.’>0,
’nZrt IlyhA bwd wfy nfshA tsA&l En Alnwm wsrh.’>0,
’lmA*A ynAm AlnAs.’>0,
’OlA ystTyE Almr@ On ybqY mstyqZA.’>0,
’nZrt mn AlnAf*p.’>0,
’kAn Alqmr yskb Dw@A rA}EA.’>0,
’qAlt.’>0,
’lmA*A yshr Alqmr kl AllyAly.’>0,
’wlmA lm tjd OHdA mstyqZA fy mvl h*h AlsAEp.’>0,
’OzAHt AlgTA@ EnhA.’>0,
’bhdw@ wgAdrt Algrfp.’>0,
’fqd $Ert OnhA bHAjp IlY qlyl mn AlmA@.’>0,
’tsllt ElY r&ws OSAbE qdmyhA.’>0,
’HtY lA tzEj OHdA.’>0,
’lknhA dh$t Hyn wjdt OmhA jAlsp tnsj AlSwf.’>0,
’fsOlthA.’>0,
’mAmA.’>0,
’lmA*A lm tnAmy bEd.’>0,
’qAlt AlOm.’>0,
’$Ert Onny lA OstTyE Alnwm.’>0,
’fjlst lOkml h*h Alknzp.’>0,
’EAdt Ebyr IlY frA$hA wbdOt tklm nfshA.’>0,
’Alqmr yshr.’>0,
’yskb Dw@h lyr$d AlnAs fy Aldrwb AlbEydp.’>0,
’wyslyhm lynswA tEbhm.’>0,
’Omy tshr ltnsj AlSwf wtmnHnA Aldf@.’>0,
’wOnA Oshr wHydp Ofkr fy h*h AlHyAp Aljmylp.’>0,
’nAmt Ebyr fy sAEp mtOxrp.’>0,
’nAmt nwmA EmyqA wHlmt OHlAmA jmylp.’>0,
’wfy AlSabAH jA@t AlOm wmsHt bydhA AllTyfp wjha Ebyr.’>0,
’ftHt Ebyr EynyhA.’>0,
’kAnt OmuhA tbtsm lhA wtdEwhA lttnAwl AlfTwr.’>0,
’fAlwqt ymr bsrEp.’>0,
’nhDt Ebyr.’>0,
’nZrt mn AlnAf*p.’>0,
’kAnt Algywm tgTy wjh AlsmA@.’>0,
’ybdw On Al$tA@ yTrq AlObwAb.’>0,
’t*krt lylp AlbArHp.’>0,
’AlsmA@ AlSAfyp bnjwmhA AllAmEp.’>0,
’wqmrhA AlwAsE Almnyr.’>0,
’qAlt AlOm.’>0,
’AlTqs tgyr bsrEp.’>0,
’Inh ymyl IlY Albrwdp.’>0,
’lA txrjy qbl On trtdy knztk Aljdydp.’>0,
’Erft Ebyr On OmhA shrt Allylp AlmADyp mn Ojl InjAz h*h  Alknzp.’>0,
’km kAnt Alknzp jmylp.’>0,
’lbst Ebyr knzthA Aljdydp.’>0,
’nZrt fy AlmrAp.’>0,
’Abtsmt wtmtmt.’>0,
’km Ont jmylp yA knzty.’>0,
’lknhA lm tns On t$kr OmhA.’>0,
’O$rq wjh AlOm why trY AbnthA trtdy Alknzp.’>0,
’wfy Almdrsp bdA AltlAmy* yzhwn bmlAbshm AlSwfyp Aljdydp.’>0,
’lm tql Ebyr h*h Almrp.’>0,
```

```
'km Ont jmylp yA knzty.'>0,
'bl qAlt.'>0,
'km hy jmylp Oydy AlOmhAt Alty HAkt h*h AlknzAt.'>0,
'wOdrkt On kl AlOmhAt yshrn mE Alqmr ySnEn $y}A jmylA.'>0,
'kAnt ndY tnZmu wqthA fy OyAm AlETl.'>0,
'tHDr wAjbAthA Almdrsyp.'>0,
'wtsAEd OmhA fy bED AlOEmAl AlbsyTp.'>0,
'vm tlEb qlylA mn Alwqt.'>0,
'fy OHd AlOyAm nsyt On tHfZ drwshA.'>0,
'wtktb wZA}fhA.'>0,
'wrAHt tlEb mE qThA flfl.'>0,
'wfy Alywm AltAly sOlt AlmElmp ndY En wZyfthA.'>0,
'wqft ndY xjlp wqAlt.'>0,
'nsyt On OktbhA.'>0,
'qAlt AlmElmp.'>0,
'hl yrDyk On thmly wAjbk yA ndY.'>0,
'Smtt ndY wlm tjb.'>0,
'wEndmA EAdt IlY Albyt kAnt Hzynp.'>0,
'tnAwlt TEAmhA wjlst tqrO drwshA bSmt.'>0,
'Aqtrb mnhA flfl whw ymw@.'>0,
'qAlt ndY lflfl.'>0,
'A*hbo wAbHvo En krp tlhw bhA.'>0,
'OmA OnA fOryd On OqrO lOSbH mjthdp wtHbny mElmty.'>0,
'Hzn flfl.'>0,
'wAnzwY bEydA yfkr.'>0,
'lmA*A Trdtny ndY.'>0,
'wErf Onh yjb Ono ytrkhA bED Alwqt.'>0,
'ltktb bwZA}fhA.'>0,
'wElyh On yEml hw OyDA.'>0,
'wmn* *lk Alywm tElm flfl OlA ytrk AlH$rAt AlDArp wAlf}rAn  Alm&*yp thrb mn mxAlbh OmA ndY fkAnt tmsH $Erh AlnAEm  brfq.'>0,
'wtHmlh IlY AlHdyqp.'>0,
'xlAl AstrAHthA.'>0,
'wtlEb mEh bsrwr.'>0,
'wEdp OsrAb mn AlfrA$ AllTyf.'>0,
'kAnt jmAEAt AlOzhAr wAlwrd tstmE lHkAyAt AlfrA$.'>0,
'wtn$r ETrhA tEbyrA En frHthA bSdAqp AlfrA$ wHyn y$td  AlHr.'>0,
'kAnt AlfrA$At tTyr wtHT.'>0,
'trfrf bOjnHthA.'>0,
'tlTf Aljw.'>0,
'ltxff mn qsAwp AlHr En OSdqA}hA.'>0,
'wI*A jA@ Allyl.'>0,
'wtEbt AlfrA$At tnAm fy OHDAn Alwrd wAlOzhAr bhdw@ mE  yrqAthA AlSgyrAt.'>0,
'*At ywm tErD Hql mjAwr lHryq.'>0,
'wAmtd Allhb IlY Hql AlOSdqA@.'>0,
'bsrEp smE AljmyE xbr AlHryq.'>0,
'OsrEt OsrAb AlfrA$.'>0,
'w$klt HAjzA mn OjsAdhA lHmAyp AlOSdqA@.'>0,
'kAn Allhb ylfH wjh Alwrd wAlOzhAr wAlE$b.'>0,
'AHtrq kvyr mn AlfrA$ qbl On ytlA$Y Allhb.'>0,
'wynTf} AlHryq OmA Alwrd wAlOzhAr.'>0,
'fqd AHtDnt AlyrqAt AlSgyrAt HtY OSbHn frA$A ymlO AlHql  sEAdp wjmAlA.'>0,
'dxl AlmElmu IlY AlSaf fjOp.'>0,
'Smt AljmyE wsAd hdw@ tAm.'>0,
'kAn wAjmA wbdt ElAmAt AlgDb wAlAnzEAj ElY wjhh.'>0,
'nZr fy wjwh AltlAmy* wAHdA wAHdA wkOnh ybHv En $y@ ODAEh.'>0,
'tnfs bEmq wqAl bSwt y$bh Alhms.'>0,
'lqd ETlwA AldrAsp
fhm AljmyE On AlIsrA}ylyyn OmrwA bIglAq Almdrsp.'>0,
'wbEd lHZAt qAl bSwt wADH wqwy.'>0,
'sntAbE Aldrws fy Albywt
wqf wlyd wqAl.'>0,
'ldynA grfp kbyrp.'>0,
'sOTlb mn Oby On ysmH lnA bOn ndrs fyhA.'>0,
'xrj AltlAmy* mn AlSfwf.'>0,
'vm gAdrwA bhw Almdrsp kAn jnwd AlEdw yml&wn Al$ArE Alr} ysy.'>0,
'wEnd mdAxl AlOzqp AlmtqATEp mE h*A Al$ArE.'>0,
'kAnt bED AlOmhAt yntZrn OTfAlhn.'>0,
'lm ytwjh AlOwlAd IlY mnAzlhm.'>0,
'bl twzEwA IlY mjmwEAt.'>0,
'kl mjmwEp Atjht IlY zqAq frEy mtslHyn bAlHjArp wAlmqAlE  wAlzjAjAt.'>0,
'mn Oyn Zhrt kl h*h AlO$yA@.'>0,
'lqd kAnwA yxfwnhA fy mHAfZhm.'>0,
```

```
’wtHt AlvyAb.’>0,
’mr OHdu AlmElmyn wrOY mA yHdv.’>0,
’Omra AlOTfAl AlSgAr.’>0,
’A*hbwA IlY Albyt OyhA AlSgAr
rda Tfl.’>0,
’nHn kbAr
Abtsm AlmElm wtAbE Tryqh.’>0,
’kAn yErf On mErkp stHdv wkAn msrwrA.’>0,
’jdty AsmhA AlHAjp Amnp.’>0,
’kl AlnAs fy jbAlyA yErfwnhA.’>0,
’why tErf kl OhAly jbAlyA.’>0,
’jdty AlHAjp Amnp tHb kl AlnAs fy jbAlyA.’>0,
’whm yHbwnhA.’>0,
’kl Al$bAb wAlOTfAl.’>0,
’wHtY AlrjAl ynAdwnhA.’>0,
’jdty lOnhA sAEdt OmhAthm fy OvnA@ wlAdthm.’>0,
’why Owl mano Hamala OjsAdhm AlSgyrp fy Owl lHZp mino  HyAthm.’>0,
’why Owl mano tTlq zgrwdpa frH.’>0,
’wdA}mA nrAhA mbtsmp lm tbki mrp fy HyAthA fhy tzgrd End  AlwlAdp lOn qAdmA jdydA Hl fy jbAlyA ftqwl.’>0,
’AlHmd Allh zAd EddnA wAHdA.’>0,
’wtzgrd EndmA ymwt wAHd mn Almxym $hydA mn Ojl AlwTn  ftqwl.’>0,
’AlHmd llh.’>0,
’lqd SEd wAHd mnA IlY AlsmA@ wI*A sOlhA OHdu AlOwlAd.’>0,
’mA*A yfEl Al$hyd fy AlsmA@ yA jdty.’>0,
’tqwl lh.’>0,
’AnZr IlY h*h Alnjwm Alkvyrp.’>0,
’InhA OrwAH Al$hdA@ tDy@ lnA OyAmnA.’>0,
’qlt lhA *At ywm.’>0,
’Oryd On OSEd IlY hnAk lOSbH njmA.’>0,
’mA*A OEml.’>0,
’nZrt Ily wkAnt tSnE mn AlxyTAn mqlAEA.’>0,
’qAlt.’>0,
’h*A mqlAE sOdrbk kyf tDrb bh mn srq OrDnA wqtl ObAk
qlt.’>0,
’hl SEda Oby IlY Alnjwm.’>0,
’hzt rOshA qA}lp.’>0,
’nEm.’>0,
’kAnt ydAhA tEmlAn bn$AT wmhArp.’>0,
’lqd Onht AlmqlAE.’>0,
’kAn jmylA fqd rsmt bAlxywT Almlwnp AlElm AlwTny.’>0,
’knt Orgb On tETyny h*A AlmqlAE lODrb bh jnwd AlEdw.’>0,
’lkn ybdw OnhA wEdt IHdY Alfrq AlDArbp bEdd mn h*A AlslAH.’>0,
’fqd sHbt mn tHt AlfirA$ EddA kbyrA mn AlmqAlyE Alty  HAkthA.’>0,
’xbOthA fy SdrhA wxrjt bsrEp gdA sylby OhAly jbAlyA ndA@  AlAntfADp bAlIDrAb AlEAm.’>0,
’wsySEd bEDhm IlY AlsmA@ wstzgrd jdty.’>0,
’nZrt AlHAjp Amnp IlY Swr ObnA}hA AlvlAv mElqp fy Sdr Albyt  wqAlt.’>0,
’AlHmd llh Al*y $rfny bAst$hAdhmØ´ kAn Tfl Sgyr ynAm fwq  Alsryr Alkbyr.’>0,
’wqd lufa jsdh bAlkwfyp AlflsTynyp.’>0,
’Inh Abn wldhA Hsn Al*y Ast$hd qbl wlAdp Tflh bvlAvp OyAm.’>0,
’Osmth Aljdp Amnp Hsn wqAlt HynhA.’>0,
’AlHmd Allh
lqd EAd Hsn.’>0,
’Aqtrbt Aljdp Amnp mn HfydhA Aljdyd wElY wjhhA AbtsAmp  jmylp.’>0,
’qAlt bSwt hAms.’>0,
’nmo yAHbyby nmo lqd shr Obwk lynAm AlOTfAl namo yA Hbyby  stkbr wtshr mvl Obyk.’>0,
’nmo hnAk mn yshr AlAn mn Ojlk.’>0,
’stnhD ywmA Hyn t$rq $ms AlwTn.’>0,
’wtkwn kbyrA OmA OnA fsO*hb AlAn.’>0,
’rbmA ln OrjE.’>0,
’stkbr wtgny blAdy.’>0,
’blAdy.’>0,
’xrjt Aljdp Amnp bEd On Oxft $y}A fy SdrhA wkAn Hsn ynAm  bhdw@.’>0,
’jls n$wAn.’>0,
’jAnb AlnAf*p Almglqp.’>0,
’ylEb bOlEAbh.’>0,
’Inh lA ystTyE Alxrwj IlY Al$ArE.’>0,
’fAlmTr yhTl fy AlxArj.’>0,
’Akt$fa n$wAnu $y}A OEjbhu.’>0,
’Akt$fa Swt HbAt AlmTr AlmtsAqTp brtAbp wkOnhA tgny.’>0,
’wrAH yuSgy bfrH IlY hsys AlmA@ AlmnsAb mn AlOsTHp OyDA.’>0,
’fjOp qTEt Elyh OSwAt qwyp ISgA@ahu.’>0,
’wkAnt gyr mOlwfp ln$wAn.’>0,
```

```
'fxAf wrkD IlY AlmTbx Hyv kAnt Omuhu tHDr AlTEAm.'>0,
'wSAH.'>0,
'mAmA.'>0,
'mAmA.'>0,
'mAh*h AlOSwAt Alqwyp.'>0,
'OnA xA}f.'>0,
'lkn AbtsAmpa Omih hdaOt mn xwfh wADTrAbh
lA txf yAbny.'>0,
'h*h OSwAt AlrEd.'>0,
'mAhw AlrEd.'>0,
'AlrEd OSwAtu Algywm fy AlsmA@.'>0,
'wlmA*A tSrx Algywm bOSwAt mxyfp.'>0,
'hl tt$Ajr Algywmu yA Omy.'>0,
'nEm.'>0,
'InhA tt$Ajr qlylA.'>0,
'wlknhA txjl mn tSrfhA ftSmt wtnzl mTrA.'>0,
'mAmA.'>0,
'hl AlmTr hw dmwE AlgymAt.'>0,
'TbEA Inh dmwE AlgymAt *rfthA ndmA ElY Al$jAr.'>0,
'mAmA mn Oyn tOty AlgymAt.'>0,
'mn AlbHr yA bny.'>0,
'EAd n$wAn IlY jAnb AlnAf*p.'>0,
'wOSgY TwylA IlY HbAt AlmTr.'>0,
'why trqS ElY AlsTwH.'>0,
'wfy Al$ArE.'>0,
'wkAn ysmE.'>0,
'OHyAnA.'>0,
'Swt AlrEd.'>0,
'fyDHk lOnh yErf On AlgymAt tt$Ajr qlylA.'>0,
'wOn dmwEhA tsqy AlOrD wAlOzhAr wAlESAfyr.'>0,
'*At mrp HAwlt On Omsk ESfwrA HyA.'>0,
'ljOt IlY AlHylp kmA yfEl kl AlOwlAd.'>0,
'jhztu Hfrp tsE ESfwrA kbyrA.'>0,
'wOHDrt qTEp mn AlSxr ElY $kl rqAqp.'>0,
'vm OsndthA bAlEydAn b$kl mnAsb.'>0,
'wlm Onsa Ono Ovbt dwdp Hyp mn dydAn AlOrD.'>0,
'vm mkvt.'>0,
'dwn HrAk.'>0,
'bEydA En AlHfrp.'>0,
'OrAqb AlESAfyr trwHu wtjy@u.'>0,
'tHT hnA.'>0,
'tnTu hnAk bAHvp En TEAmhA wTEAm OwlAdhA.'>0,
'wlm OTli Almkwv.'>0,
'lOn ESfwrA jA}EA.'>0,
'kAn qd $Ahd dwdp AlOrD ttHrku dAxl AlHfrp.'>0,
'fAnqDa ElY Aldwdp.'>0,
'wlm ydri Onh wqE fy Alfx.'>0,
'I* OTbqt Elyh rqAqp AlSxr wHbsth fy AlHfrp.'>0,
'tsArEt dqAt qlby Hyn $Ahdtu AlESfwr yqE fy AlmSydp Alty  rtbthA lh.'>0,
'wOsrEt Ilyh.'>0,
'lm Okno frHA.'>0,
'bl kntu mDTrbA.'>0,
'xA}fA.'>0,
'lA OErf lmA*A.'>0,
'txylt nfsy ESfwrA wqE fy mSydp.'>0,
'wlA ystTyE Alxrwj mnhA.'>0,
'smEtu DrbAti jnAHy AlESfwr dAxla AlHfrp.'>0,
'kAnt ydAy trtjfAn Hyn bdOtu Emlyp AlqbD ElY AlESfwr.'>0,
'b*ltu jhdA HtY lA yflt mny.'>0,
'knt Oryd On yrY rfAqy AlESfwr fy ydy.'>0,
'lOvbt lhm Onny SyAd mAhr mvl Oy wAHd mnhm.'>0,
'Hfrt Hfrp Sgyrp jAnb AlHfrp Alkbyrp.'>0,
'wOdxlt ydy.'>0,
'bl tsllt OSAbE kfy AlSgyrp bxwf kbyr.'>0,
'wkOnny sOqbD ElY jmrAt mn nAr.'>0,
'hAhy OSAbEy tlAms Alry$ AlnAEm.'>0,
'bdO AlESfwr ydwr.'>0,
'yhrb mn OSAbEy.'>0,
'why tlAHqh.'>0,
'HtY Omskt bh.'>0,
'lm ystslm AlESfwr.'>0,
'kAn yntfD bqwp.'>0,
```

```
’fOHTth bkltA ydy wSrxt bSwt EAl.’>0,
’ESfwr.’>0,
’ESfwr.’>0,
’lqd ASTdt ESfwrA.’>0,
’lm ysmEny OHd.’>0,
’bdOt Odwr fy mkAny wAlESfwr ytxbT byn ydy AlmHkmtyn Elyh.’>0,
’kAnt AlESAfyr AlOxrY tTyr mn $jrp IlY OxrY tqfz fwq  AlOrD.’>0,
’tft$ En g*A}hA.’>0,
’HynhA $Ertu Onny fEltu $y}A b$EA.’>0,
’fArtjfto ydAy b$dp.’>0,
’wArtxto OSAbEy.’>0,
’wrOyt ESfwry ymDy kshm fy AlfDA@.’>0,
’mAzlt O*kr *lk klmA rOyt ESfwrA fOhms h*A hw ESfwry.’>0,
’kAn frAs ynAmu bEmq Hyn gAdrt Omh Albyt.’>0,
’lt$try AlHAjAt AlDrwryp.’>0,
’kEAdthA kl ywm.’>0,
’wtEwd qbl On ystyqZ.’>0,
’lkn frAs lmo yTli Alnwm h*A AlSbAH.’>0,
’fqd AstyqZ bEd Ono gAdrto Omh bqlyl.’>0,
’nZr fy OrjA@ Algrfp flm yjdo OHdA.’>0,
’frk Eynyh.’>0,
’OnSta qlylA.’>0,
’rbmA ysmE OSwAt AlOTbAq Alty tgslhA Omuh kl SbAH.’>0,
’lkn lA Swt yOty mn nAHyp AlmTbx.’>0,
’HtY AlqT Al*y ylEb mEh kl ywm gyr mwjwd.’>0,
’SAH frAs.’>0,
’mAmA.’>0,
’mAmA lknh lm ysmEo jwAbA.’>0,
’nAdY bSwt OqwY.’>0,
’lknh lmo ysmEo OHdA yrd Elyh.’>0,
’fbdO ybky bSwt qwy ltsmEh Omuh.’>0,
’dAr fy Algrfp.’>0,
’wkOnh ybHv En $y@ ODAEh.’>0,
’fjOp $Ahd Swrth fy AlmrAp Alkbyrp Almwjwdp OmAm AlxzAnp.’>0,
’$Ahd Swrth tbky mvlh.’>0,
’dh$ mn wjwd wld fy AlmrAp.’>0,
’ftwqf En AlbkA@.’>0,
’wAqtrb mn AlmrAp.’>0,
’wqAl llwld Al*y fy AlmrAp.’>0,
’hl trktk Omuk mvly.’>0,
’$Ahd kyf tHrkt $ftA Alwld fy AlmrAp.’>0,
’lknh lmo ysmEo Swth.’>0,
’fEAd yqwl lh.’>0,
’hl ODEt Swtk.’>0,
’wlm tjdoh.’>0,
’wkAn yrY $fty Alwld ttHrkAn fy kl mrp yHdvh.’>0,
’nsy frAs gyAba Oumih.’>0,
’wrAH yHdv Tfla AlmrAp wkAn AlTfl yHdvh dwn Swt.’>0,
’wklmA Aqtrb mn AlmrAp.’>0,
’kAn y$Ahd Tfl AlmrAp yqtrb mnh Okvr.’>0,
’wHyn wDE kfah ElY wjh AlmrAp.’>0,
’kAn Alwld yDE kfh fwq kf frAs OyDA.’>0,
’wI*A DHk frAs kAn Alwld fy AlmrAp yDHk OyDA.’>0,
’EAdt AlOm mn Alswq.’>0,
’daxalato bhdw@ HtY lA twqZ AbnhA.’>0,
’fqd Znt Onh mAzAl ygTa fy nwm Emyq.’>0,
’wmA Ino dxlto HtY smEto Swt frAs wDHkAth wkOnh yHdvu OHdA  mA.’>0,
’wAEtqdto On ObAh qd EAd lOmr mA fwjdh mstyqZA.’>0,
’lknhA fwj}t EndmA rOt AbnhA ylAEb Swrth fy AlmrAp.’>0,
’wyDHk fqAlt.’>0,
’hA OnA qd Edt.’>0,
’tEAl wAnZr mA*A OHDrt lk.’>0,
’fqAl frAs dwn On yltft.’>0,
’lys AlAn.’>0,
’OnA OlEb mE Sdyqy.’>0,
’Aktfti AlOmu bAbtsAmp jmylp.’>0,
’trktohu ylEbu mE Swrthi w*hbto IlY OEmAlhA.’>0,
’wykrr nSyHth Alty HfZnAhA En Zhr qlb.’>0,
’A$trwA O$yA@ mfydp.’>0,
’wkAn kl mnA ysEd jdA EndmA yDE Alnqwd fy jybh.’>0,
’wyrsm fy *hnh mgAmrp Sgyrp tnAsb qymp h*h AlqTE.’>0,
’sO$try AlTbA$yr Almlwnp.’>0,
```

```
'wOqdmhA llmElmp.'>0,
'sO$try SHnA mn Alfwl mn Oby mHmwd.'>0,
'lkn Oxy wA}li kAn ysrE IlY Almktbp Alx$byp.'>0,
'Alty wDE.'>0,
'ElY Osfl rf mnhA.'>0,
'HSAlth Alty OhdthA Ilyh OmunA.'>0,
'fnsmE Swt AlqTE Alnqdyp AlmEdnyp AlmtsAqTp fy AlHSAlp.'>0,
'wkAn h*A yvyrny HqA.'>0,
'wOtsA@l.'>0,
'lima ystTyE wA}l AlSgyr Ono ywfr nqwdh.'>0,
'wlA tgryh bAl$rA@ mn dkAn AlbqAl.'>0,
'wkvyrA mA $Ertu bAlHsd wAlIEjAb bqdrth ElY AlSbr btwfyr  xrjyth bynmA.'>0,
'nHn AlkbAr.'>0,
'lA nstTyE mqAwmp IgrA@ AlHlwY All*y*p.'>0,
'wAlO$yA@ Aljmylp Alty tlmEu xlfa zjAj AlmEArD AltjAryp.'>0,
'wqrrtu mrp Ono O$try HSAlp wqlt fy nfsy.'>0,
'sODEu fyhA kla mA OHSlu Elyh mn nqwd mno Oby wOmy wjdty.'>0,
'wlknny lm OstTEo $rA@ AlHSAlp.'>0,
'fqd tbxrt nqwdy qbl On Odxl bAHp Almdrsp.'>0,
'lOn AlbxAr AlmtSAEd mn Erbp AlEm Oby mHmwd.'>0,
'bA}E Alfwl Hrk Alrgbp dAxly.'>0,
'On Ot*wq TEm Alfwl mE AlHmD.'>0,
'w$Ert bAlndm wlkn bEd fwAt AlOwAn.'>0,
'w$glny *lk kvyrA.'>0,
'HtY Ony $rdtu OvnA@ $rH Aldrs wnbhny AlmElm.'>0,
'mAlka yArbyE.'>0,
'hl t$Eru b$y@.'>0,
'mA*A y$glu *hnk h*A Alywm.'>0,
'w$Ertu bxjl $dyd.'>0,
'wHsbt On kl zmlA}y ynZrwn Ily.'>0,
'wfy Albyt qlt lOmy.'>0,
'mAmA.'>0,
'Oryd HSAlp kHSAlp wA}l.'>0,
'lAHZt Omy ElAmAt AlAnzEAj bAdyp ElY wjhy fqAlt.'>0,
'hl h*A mA y$gl bAlk wyzEjk.'>0,
'qlt.'>0,
'sOHAwl On Owfr mvl wA}l.'>0,
'Abtsmt Omy qA}lp.'>0,
'lA tqlqo.'>0,
'sykwn lk HSAlp h*A Alywm wqbl mgyb Al$ms.'>0,
'fElA.'>0,
'lqd brto Omy bwEdhA.'>0,
'wA$trto ly HSAlp t$bh HSAlp wA}l.'>0,
'lknhA txtlf bAllwn.'>0,
'Hmltu AlHSAlp bydyn mrtE$tyn.'>0,
'wkOny OHml knzA.'>0,
'wdArt fy *hny OHlAm kvyrp.'>0,
'stmtl} HSAlty bAlnqwd.'>0,
'wsO$try mA O$thy mn AlOlEAb wAlHlwY.'>0,
'sO$Ark fy AlrHlp Almdrsyp dwn On Oklf Oby dfE Almblg  AlmTlwb.'>0,
'sOSlH drAjty AlmETlp.'>0,
'wOlEb bhA fy OwqAt frAgy.'>0,
'wtwAlti AlOfkAr wAlOHlAm.'>0,
'kAnt Omy trAqbu AnfEAlAty AlbAdyp ElY wjhy wAlAbtsAmp tDy@  wjhhA.'>0,
'qAlt why tETyny Edp qTE mn Alnqwd AlmEdnyp.'>0,
'DEo h*h Alnqwd fy HSAltk Aljdydp.'>0,
'wHAwlo On tDyf IlyhA kl SbAH.'>0,
'OsqTt AlqTE Alnqdyp dAxl HSAlty qTEp qTEp bynmA kAnt tdwr  Swr kvyrp fy mxylty.'>0,
'drAjty Alty tntZr AlISlAH.'>0,
'AlrHlp Almdrsyp.'>0,
'Erbp Alfwl wAlbxAr AlmtSAEd mnhA.'>0,
'AlqSS AlmSwrp fy wAjhp Almktbp Alqrybp mn bytnA.'>0,
'AxtlTt kl h*h AlSwr wOnA ODE HSAlty Aljdydp IlY jAnb HSAlp  Oxy wA}l.'>0,
'mA*A yqwl AlbHr
wqft SbA ElY $AT} AlbHr.'>0,
'nZrt IlY AlbEyd Hyv yltqy AlbHr bxT AlOfq.'>0,
'kAn AlmnZr mdh$A Hrkt EynyhA fy jmyE AljhAt rOt zwrqA  bEydA.'>0,
'kAn ybdw SgyrA jdA.'>0,
'tmnt fy nfshA lw OnhA trkb h*A Alzwrq.'>0,
'wtjwb OnHA@ AlbHr AlrHb.'>0,
'wt*krt OnhA lm ttElm AlsbAHp bEd.'>0,
'fI*A sqTt fy AlmA@ mA*A yjry lhA.'>0,
```

’lAmst mwjp qdmy SbA blTf.’>0,
’wjElthA tntbh mn $rwdhA.’>0,
’wjlst ky trAqb md Almwj wjzrh.’>0,
’Aqtrbt Okvr Hyv tlTmhA AlmwjAt AlmtlAHqp.’>0,
’Odh$hA hsys Almwj fwq Alrml fy tqdmh wtrAjEh.’>0,
’wtsA@lt.’>0,
’mA*A yqwl AlbHr llrml.’>0,
’wmA*A yqwl Alrml lh.’>0,
’wxTr lhA On tktb AsmhA fy dftr Al$AT}.’>0,
’ktbt SbA jA@t yd AlbHr wmHthA.’>0,
’OEAdt SbA AlktAbp.’>0,
’Amtdto ydu AlbHr mrp OxrY wmHthA.’>0,
’lEbt SbA mE AlbHr TwylA.’>0,
’banato bytA kbyrA mn Alrml.’>0,
’wjElt lh swrA mn Alrml OyDA lkn AlbHr Orsl mwjp kbyrp hdmt  lhA Albyt wAlswr.’>0,
’lm tstslm SbA bl OEAdt bnA@ Albyt Alrmly wAlswr OyDA.’>0,
’lkn h*h Almrp.’>0,
’fy mkAn bEyd En yd AlbHr wEndmA Onht bnA@ bythA Alrmly nfDt  ydyhA mn AvAr Alrml wqAlt mwjhp klAmhA IlY AlbHr.’>0,
’hyh.’>0,
’lA ymknk hdm byty h*h Almrp kAnt AlOmwAj trkD wtrkD.’>0,
’lknhA lm tSl IlY Albyt Al*y bnth SbA.’>0,
’kAnt SbA sEydp lqd lEbt mE AlbHr TwylA.’>0,
’kAn OsAmp yqfz whw ytrnm trllA.’>0,
’trllA.’>0,
’trllA Hyn smE hdyr TA}rp fy AlsmA@.’>0,
’wqf wrfE rOsh IlY OElY.’>0,
’mHAwlA On yrY h*h AlTA}rp.’>0,
’wt*kr s&Al mElimh lltlAmy* fy AlSf.’>0,
’mA*A tHb On tkwn fy Almstqbl.’>0,
’wHynhA fkr OsAmp.’>0,
’hl Oqwl.’>0,
’OHb On Okwn mElmA.’>0,
’Ow lAEbA ryADyA.’>0,
’t*kr SyAH AltlAmy* OnA OHb On Okwn sA}q syArp OnA Oryd On  Okwn $rTy mrwr.’>0,
’wOnA sOkwn fnAnA.’>0,
’OnA.’>0,
’OnA.’>0,
’kAnt AlTA}rp qd gAbt En Eyny OsAmp wSwt mHrkhA tlA$Y  OyDA.’>0,
’lkn mAzAl SdY *lk AlSwt fy *hn OsAmp.’>0,
’SAH OsAmp bSwt EAl.’>0,
’OHb On Okwn TyArA.’>0,
’wDAE Swth fy AlfDA@ mvl Swt AlTA}rp.’>0,
’wbdO xyAlh ySwr lh nfsh TyArA yqwd TA}rp Hrbyp tHmy smA@  AlwTn.’>0,
’vm TyArA yqwd TA}rp zAxrp bAlrkAb.’>0,
’kAn yqf wyrAqb AlsmA@ $Ahd gywmA mtfrqp wbDE HmAmAt tTyr fy  srb wAHd.’>0,
’EAd yqfz frHA whw yrdd.’>0,
’trllA.’>0,
’tlAllA.’>0,
’OnA TyAr.’>0,
’OnA TyAr.’>0,
’dh$ AlOwlAd Hyn ElmwA On mAhr symlO slth bAlkrz.’>0,
’f$jr Alkrz lA yvmr fy Al$tA@.’>0,
’qAlt swsn.’>0,
’mn Oyn stmlO sltk bAlkrz.’>0,
’O$Ar mAhr bydh IlY AlsmA@.’>0,
’AnZrwA.’>0,
’h*h $jrp qws qzH tHml krzA kvyrA.’>0,
’nZr AlOwlAd IlY Aljhp Alty O$Ar IlyhA mAhr.’>0,
’kAn qws qzH bOlwAnh Almmyzp ybdw rA}EA.’>0,
’qAl mjd.’>0,
’fy bstAn qws qzH O$jAr tHml brtqAlA nADjA.’>0,
’Sfqto njwd wSAHto bSwt EAl.’>0,
’mA Ojml h*h AlHbAl Almlwnp.’>0,
’sOxtAr AlHbl AlOSfr lOlEb lEbp nTi AlHbl
wqAl mnAr.’>0,
’OnA OrY HqlA OxDr.’>0,
’sAx* xrwfy lyrEY wjbp mn AlE$b AlTry.’>0,
’qAlt tymA@.’>0,
’Inh qlmy AlOzrq.’>0,
’SEd lylwn AlsmA@.’>0,
’OmA fATmp.’>0,
’fqd t*krt On OmhA Tlbt mnhA On t$try OqrAS nyl ltjml  Algsyl.’>0,

```
'fqAlt.'>0,
'sOHml Algsyl IlY bHyrp qws qzH Alnylyp.'>0,
'lySbH Algsyl zAhyA.'>0,
'kAnt Ebyr tnZr IlY qws qzH mE rfAqhA wrfyqAthA fqAlt.'>0,
'Olmo t$AhdwA OzhAr qws qzH Albnfsjyp.'>0,
'AnZrwA.'>0,
'mA OjmlhA.'>0,
'qAl OHdu AlOwlAd.'>0,
'sOrsm qws qzH fy dftry ky lA OnsAh.'>0,
'wHyn gAb qws qzH Hzn AlOwlAd kvyrA.'>0,
'qAlt swsn.'>0,
'rbmA rkb OwlAd qws qzH Zhr gymp w*hbwA lyHDrwA lnA AlhdAyA  Aljmylp.'>0,
'wtmnY mjd On yhTl AlmTr bgzArp lysqy AlHqwl wkAnt njwd tqwl  lOSdqA}hA.'>0,
'mA Ojml On OHSl ElY mndyl lOqdmh hdyp lOmy fy EydhA.'>0,
'wOxyrA qrr AlOwlAd On ylEbwA lEbp mfydp.'>0,
'qAl mAhr.'>0,
'tEAlwA yA OSdqA}y nkwn qws qzH.'>0,
'tjmE AlOwlAd frHyn.'>0,
'qAlt swsn.'>0,
'wkyf *lk.'>0,
'qAl mAhr.'>0,
'OnA Alkrz AlOHmr.'>0,
'qAl mjd.'>0,
'OnA AlbrtqAl.'>0,
'AljmyE yErfny.'>0,
'qAlt njwd.'>0,
'OnA Allymwn AlOSfr.'>0,
'tHtAjwn Ily dA}mA.'>0,
'qAl mnAr.'>0,
'OnA AlE$b AlOxDr.'>0,
'ltOti kl AlxrAf wtrEY.'>0,
'OmA tymA@ fqd Sfqt DAHkp why tqwl.'>0,
'OnA AlbHr yHbny AljmyE.'>0,
'wytmtEwn brzqty AlSAfyp.'>0,
'fy AlSyf OHml AlmrAkb AlSgyrp.'>0,
'wAlsfn Alkbyrp wysbH AlOTfAl fy myAhy mE AlOsmAk Almlwnp.'>0,
'gmzt fATmp bEynhA mbtsmp.'>0,
'sOTyr IlY AlbHr wOgmr Algsyl fy myAhh AlzrqA@ lyktsb zrqp  AlsmA@ AlSAfyp.'>0,
'Zhr qws qzH mrp vAnyp.'>0,
'kAn AlmTr yhTl mb$rA bETA@At AlHqwl.'>0,
'wkAn AlOwlAd yrqSwn tHt AlmTr.'>0]) :-
 set(firstOneOnly),
 set(softTypes),
 unset(justWords),
 unset(soft(_, 77)),
 set(translationorm).
```