Parse Trees of Arabic Sentences Using the Natural Language Toolkit

Maad Shatnawi and Boumediene Belkhouche
College of IT, UAE University, Al Ain
{shatnawi, b.belkhouche}@uaeu.ac.ae

Abstract

We develop a framework for using the Natural Language Toolkit (NLTK) to parse Quranic Arabic sentences. This framework supports the construction of a treebank for the Holy Quran. The proposed model succeeds in parsing different Quranic chapters (Suras) in addition to Modern Standard Arabic (MSA) sentences. The availability of such parser will be useful in various natural language processing applications such as machine translation, speech synthesis, and information retrieval.

Keywords: *Arabic parser, Quranic sentences parsing, NLTK.*

1. Introduction

Parsing has several applications including semantic disambiguation, machine translation, and speech synthesis. Traditional Arabic grammar is considered to be one of the historical origins of modern dependency grammar [19, 26]. There are several state-of-the-art Arabic parsers such as Bikel parser [4, 12, 20], Malt parser [16, 23, 24], Stanford parser [13, 18], and Attia's rule-based parser for Modern Standard Arabic [1, 31]. These parsers require the availability of a treebank [14].

Like all Arabic language processing tasks, Arabic parsing faces many challenges including the unique features of the language, the exceptional degree of ambiguity in its writing system, the rich morphology, the highly-complex word formation process of roots and patterns, the length of sentences, the free order of words in the sentence, and the presence of the elliptic personal pronoun "al damiir al mustatir" [25, 30]. As a result, alternative parses for a particular sentence can suggest alternative meanings [8].

One of the significant challenges and prerequisite for parsing and syntactic analysis of a highly inflected language such as Arabic is the morphological segmentation [2, 15]. In the Penn English Treebank, verbal contraction such as "weren't" are split into separate segments "were" and "n't" with a different part-of-speech tag for each of them [3]. These segments form individual units in syntactic analysis and separate leaf nodes in a syntax tree.

Modern Standard Arabic (MSA) is the literary standard Arabic currently used across the Middle East and North Africa, and one of the official six languages of the United Nations. All printed books, newspapers, magazines, official and educational documents are written in MSA. Furthermore, MSA is used in news, documentary and scientific programs across different media and online portals.

Quranic Arabic is a unique form of Arabic used in the Quran and it is the direct ancestor language of MSA. Annotating the Quran faces an extra set of challenges compared to MSA due to the fact that the text is over 1,400 years old, and the Quranic script is more varied than modern Arabic in terms of orthography, spelling and inflection. The same word is spelled in different ways in different chapters. However, the Quran is fully diacritized which reduces its ambiguity [8].

With respect to morphological segmentation, 54% of the Quran's 77,430 words require segmentation, resulting in 127,806 segments. A typical word in the Quran consists of multiple segments combined into a single whitespace-delimited word form. For example, "فاذكروني", which means "so remember me" should be segmented to four segments; "ف", "اذكر", "و", and "ني", where each segment represents an individual syntactic unit as follows: conjunction prefix, verb, subject pronoun, and object pronoun, respectively.

The computational analysis of the Quran remains an intriguing but unexplored field since little and isolated efforts have been done in this field [6]. For the young learner, this may be not just a challenging requirement, but a deterrent.

The rest of this paper is organized as follows. The next section provides a brief overview of the existing Arabic treebanks for both MSA and Quranic sentences. Section 3 presents our methodology to parse Arabic sentences through NLTK. Analysis of our results is discussed in section 4. Concluding remarks are presented in Section 5.

2. Arabic Treebanks

A treebank is a collection of manually checked syntactic analyses of sentences. Treebanks constitute important resources for building and evaluating parsers and are used in several applications such as tokenization, diacritization, part-of-speech (POS) tagging, morphological disambiguation, chunk parsing, and semantic role labeling [14].

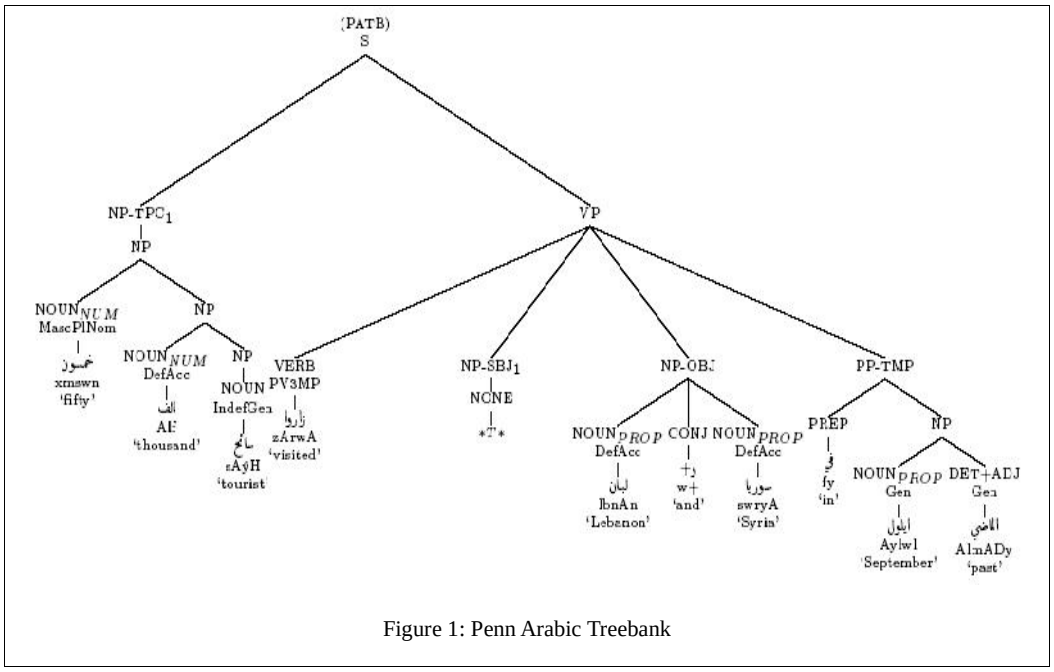

Figure 1: Penn Arabic Treebank

The three well-known Arabic treebanks for MSA are the Penn Arabic Treebank (PATB) [21, 22], the Prague Arabic Dependency Treebank (PADT) [17 28, 29], and the Columbia Arabic Treebank (CATiB) [15, 16].

The first Arabic treebank project is PATB which was started in 2001 at the Linguistic Data Consortium (LDC) and the University of Pennsylvania. The creation of PATB is considered a great achievement for Arabic Natural Language Processing. It has been a great research resource in morphological analysis, tokenization, POS tagging, disambiguation, and parsing. In addition to that, it has been used as a foundation in developing other treebanks [14].

The parsing tree in PATB for the Sentence 'Fifty thousand tourists visited Lebanon and Syria last September' (خمسون ألف سائح زاروا لبنان وسوريا في أيلول الماضي) is shown in Figure 1. The parsing trees of the same sentence in PADT and CATiB are shown in Figures 2 and 3, respectively.

The Quranic Arabic Dependency Treebank (QADT) [9] has been developed and implemented in Java at the University of Leeds, UK, as part of the Quran Corpus project [7]. It works as a treebank for Quranic sentences by providing a deep computational linguistic model based on historical traditional Arabic grammar (**اعراب القران الكريم**). The treebank is freely available under an open source license. QADT provides two levels of analysis: morphological annotation and syntactic representation. The morphological segmentation has been applied to all the 77,430 words in the Quran. The treebank also represents Quranic syntax through dependency graphs as shown in Figure 4 [9, 10]. QADT is applied to each individual Quranic verses (or part of the verse) even if that verse does not form a complete sentence unless it is joined with its neighbor verse. In addition to QADT, there are very few works done in the field of morphological analysis of Quran such as [6].

Othman et al. [25] developed a rule-based chart parser for Analyzing MSA sentences with the use of lexical semantic features to disambiguate the sentence structure and the integration with a morphological analyzer.

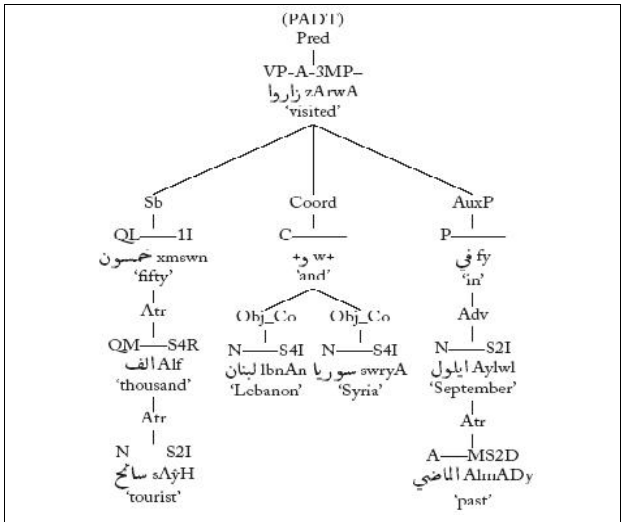

Figure 2: Prague Arabic Dependency Treebank

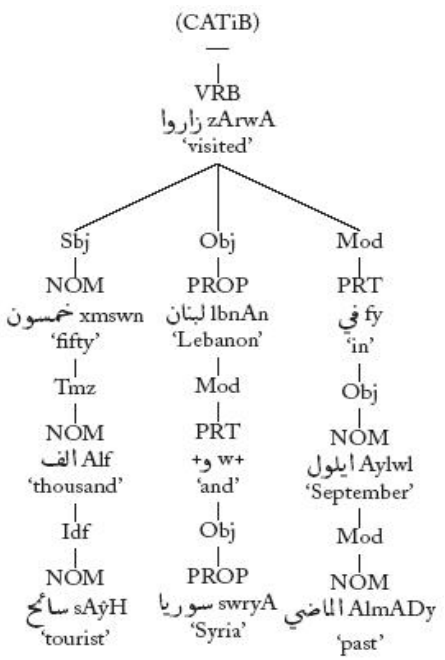

Figure 3: Columbia Arabic Treebank

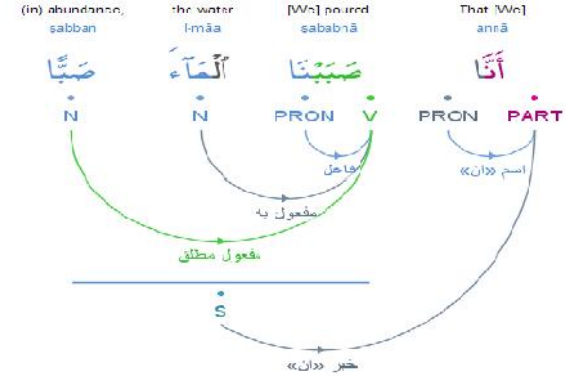

Figure 4: Parse Tree in the Quranic Treebank

3. Methodology

We apply the top-down (Recursive Descent) parsing algorithm to parse Arabic sentences through the Natural Language Toolkit (NLTK) [5]. NLTK is an open source suite of libraries and programs which can be integrated within the Python environment and then used to perform different statistical and rule-based natural language processing tasks such as POS tagging and parsing. Starting from an initial symbol S, the Recursive Descent parser applies the grammar rules forward until all the words of the given sentence are consumed. In this work, the recursive descent parser is tested and evaluated in building parse trees of Arabic sentences in general and Quranic sentences in particular. As morphological segmentation is a prerequisite for parsing, the segmentation is now done manually. Work is in under way to integrate a morphological analyzer with the parser.

Our first example demonstrates how the NLTK recursive descent parser is used to parse a MSA sentence. We use the same sentence that was parsed by the three previously mentioned Arabic parsers. The sentence is 'Fifty thousand tourists visited Lebanon and Syria last September' (خمسون ألف سائح زاروا لبنان وسوريا في أيلول الماضي). The sequence of steps to parse a text is as follows:

1. Divide the text into independent sentences.
2. Create a Python file for each sentence.
3. Perform morphological analysis.

4. Build the lexicon entries for each word in the sentence. Each word is assigned a tag:

lex1 = CFGProduction(NN, ['خمسون'])
lex2 = CFGProduction(NN, ['ألف'])
lex3 = CFGProduction(NN, ['سائح'])
lex4 = CFGProduction(VT, ['زار'])
lex5 = CFGProduction(Prn, ['و'])
lex6 = CFGProduction(NN, ['لبنان'])
lex7 = CFGProduction(CJ, ['و'])
lex8 = CFGProduction(NN, ['سوريا'])
lex9 = CFGProduction(P, ['في'])
lex10 = CFGProduction(NN, ['أيلول'])
lex11 = CFGProduction(NN, ['الماضي'])

5. Build the he grammar rules:

S_prod3 = CFGProduction(S, [NS])
VS_inv_prod1 = CFGProduction(VS, [VT, NP, CP, PP])
NS_inv_prod1 = CFGProduction(NS, [Starter, Predicate])
Starter_inv_prod1 = CFGProduction(Starter, [NP])
Predicate_inv_prod5 = CFGProduction(Predicate, [VS])
CP_inv_prod3 = CFGProduction(CP, [NP, CJ, NP])
PP_prod1 = CFGProduction(PP, [P, NP])
NP_prod1 = CFGProduction(NP, [NN])NP_prod5 = CFGProduction(NP, [Prn])
NP_prod6 = CFGProduction(NP, [NN, NP])

where S: Sentence; NS: Nominal Sentence; VS: Verbal Sentence; VT: Verb; NN: Noun, P: Preposition; PP: Prepositional Phrase; NP: Nominal Phrase; Starter: Subject of Nominal Sentence (مبتدأ); Predicate: خبر ; Prn: Pronoun; CP: Conjunction Phrase; CJ: Conjunction.

The above rules can also be represented as follows:

جملة → جملة اسمية
جملة فعلية → فعل مقطع لسمي عطف شبه جملة جار ومجرور
جملة اسمية → مبتدأ خبر
مبتدأ → مقطع لسمي
خبر → جملة فعلية
عطف → مقطع لسمي حرف عطف مقطع لسمي
شبه جملة جار ومجرور → حرف جر مقطع لسمي
مقطع لسمي → لسم مقطع لسمي
مقطع لسمي → ضمير
مقطع لسمي → لسم

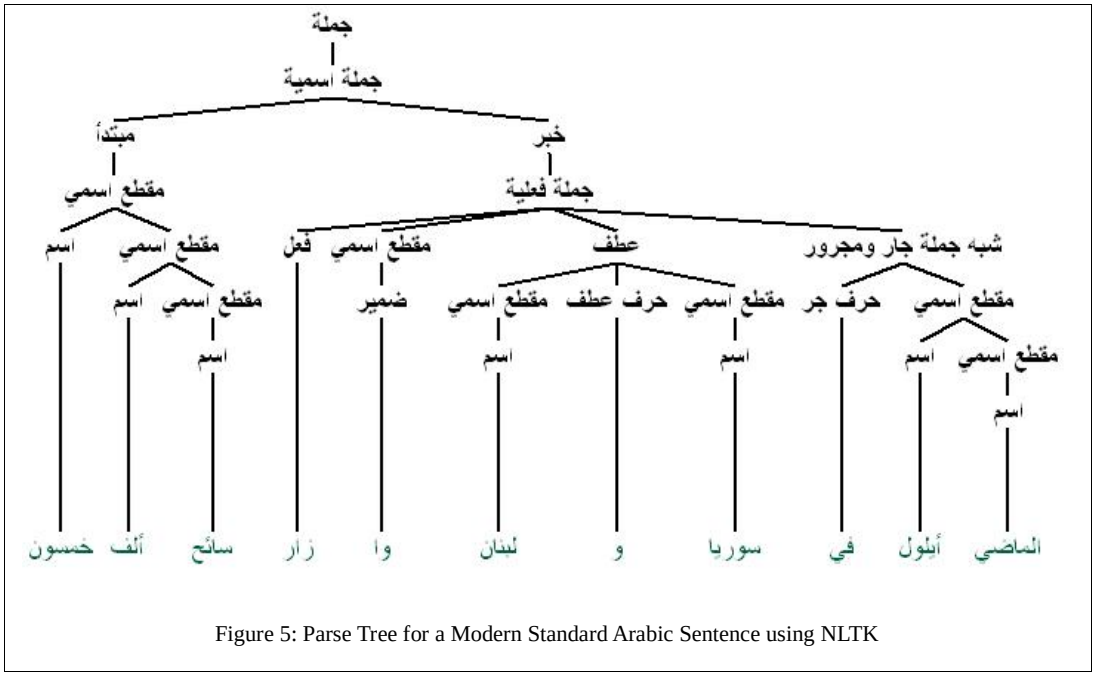

Figure 5: Parse Tree for a Modern Standard Arabic Sentence using NLTK

6. Apply the recursive descent parser (NLTK function). The generated parse tree of for the sentence is shown in Figure 5.

We now introduce a simple example to demonstrate how the NLTK recursive descent parser is used to parse Quranic sentences. The example is about parsing Verse (113:1) of the Holy Quran, the first verse of Suratul Falaq "قل أعوذ برب الفلق". The lexical entries are as follows:

lex1 = CFGProduction(VT, ['قل'])
lex2 = CFGProduction(VT, ['أعوذ'])
lex3 = CFGProduction(NN, ['رب'])
lex4 = CFGProduction(NN, ['الفلق'])
lex5 = CFGProduction(P, ['ب'])

The grammar rules are as follows:

S_prod = CFGProduction(S, [VS])
VS_inv_prod1 = CFGProduction(VS, [VT, NN, NN, PP])
VS_inv_prod2 = CFGProduction(VS, [VT, PP])
VS_inv_prod3 = CFGProduction(VS, [VT, VS])
PP_prod1 = CFGProduction(PP, [P, NP])
NP_prod1 = CFGProduction(NP, [AdP])
NP_prod2 = CFGProduction(NP, [NN])
AdP_prod = CFGProduction(AdP, [NN, NP])

Where S: Sentence; VS: Verbal Sentence; VT: Verb; NN: Noun; P: Preposition; PP: Prepositional Phrase; NP: Nominal Phrase; AdP: Additional Phrase.

The above rules can also be represented as follows:

جملة → جملة فعلية
جملة فعلية → فعل اسم اسم شبه جملة جار ومجرور
جملة فعلية → فعل شبه جملة جار ومجرور
جملة فعلية → فعل جملة فعلية
شبه جملة جار ومجرور → حرف جر مقطع اسمي
مقطع اسمي → اضافة
مقطع اسمي → اسم
اضافة → اسم مقطع اسمي

The generated parse tree of the verse is shown in Figure 6. Figure 7 presents the parse tree of Suratul Falaq (Chapter 113 of the Holy Quran) as a whole. The grammar rules of the last four Suras are presented in Appendix 1.

4. Analysis of the Results

The recursive descent parser has three main limitations [27]. The first limitation is that left-recursive productions such as "NP → NP PP" cause the parser to go in an infinite loop. Secondly, the parser blindly tries all words (lexical entries) and grammar rules that are irrelevant to the input sentence which considerably increases its time complexity. Thirdly, the algorithm does not store the successfully parsed parts which, therefore, need to be rebuilt again later in the backtracking process.

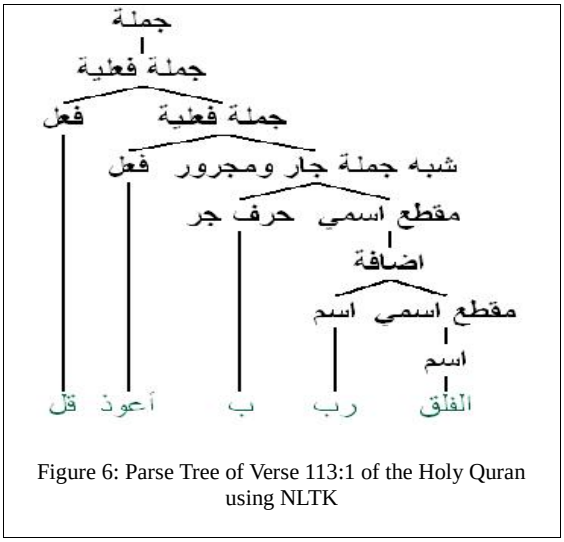

Figure 6: Parse Tree of Verse 113:1 of the Holy Quran using NLTK

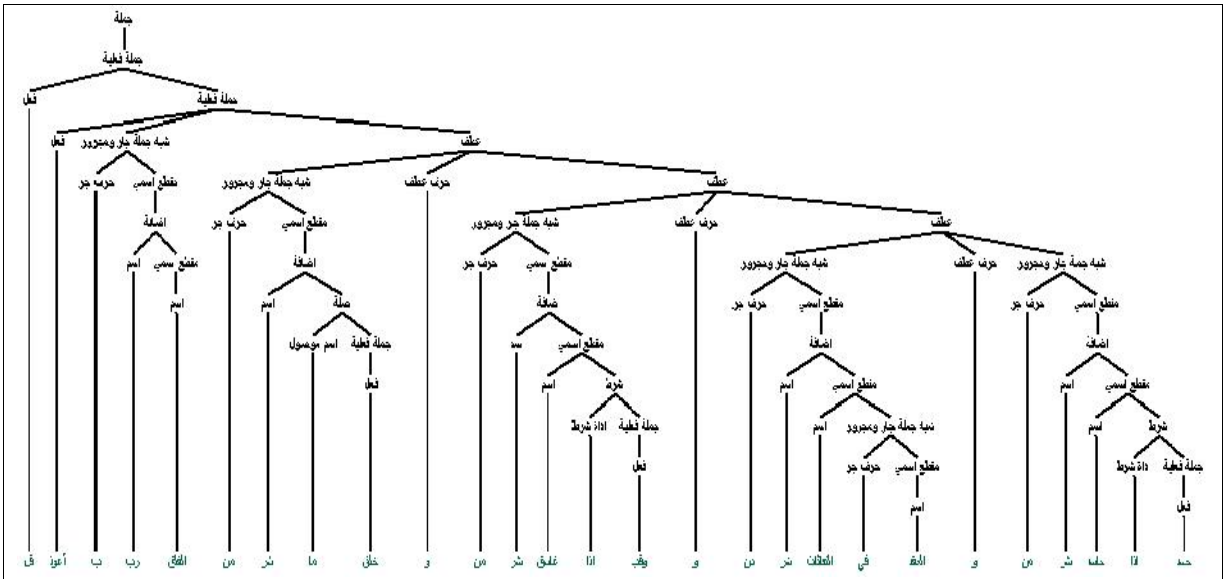

Figure 7: Parse Tree for Sura 113

For example, if the NP part in "VS → V NP" is successfully parsed, the backtracking to "VS → V NP PP" requires reparsing NP part again. As a result, the parsing processor repeats much of its work.

As the coverage of the grammar increases and the length of the input sentences grows, the number of parse trees dramatically grows. In the worst case, the recursive descent parser has an exponential time complexity as well as an exponential memory space for ambiguous context-free grammar [11]. However, for small texts like Suras 115, 114 and 115, the statistical relationship between the number of words (lexical tokens) in the sentence and the total number of nodes in the parse tree is linear. From parse tree data we collected, it is expressed as:

$$n = 3.36\, w + 0.2$$

where n is the total number of nodes in the parse tree and w is the number of words in the sentence. This expression indicates that, at least, for the examples, the complexity of the parse trees grows linearly.

5. Conclusion

We presented an approach to generate parse trees of Arabic sentences in general and Quranic sentences in particular, using the Natural Language Toolkit (NLTK). The process we defined consists of building a lexicon, a context-free grammar and invoking the NLTK recursive-descent parser. The generated parse trees can be viewed as components of a tree bank, similar to the tree banks mentioned earlier.

Our work can be extended in several directions. The integration of the parser with a morphological analyzer would further automate the process. Using diacratized sentences can dramatically decrease the semantic ambiguity and reduce the complexity of the grammar rules. Other types of parsers such as bottom-up can be also examined on Arabic sentences in a future work.

References

[1] Attia M., "An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks," *The Challenge of Arabic for NLP/MT Conference*, London, The British Computer Society, 2006.

[2] Bies A. and Maamouri M., "Penn Arabic Treebank Guidelines," 2003. http://www.ircs.upenn.edu/arabic.

[3] Bies A., Ferguson M., Katz K., and MacIntyre R., "Bracketing Guidelines for Treebank II Style, Penn Treebank Project". University of Pennsylvania, Philadelphia, 1995.

[4] Bikel D., "Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine," *International Conference on Human Language Technology Research (HLT)*, pp. 24–27, 2002. DOI: 10.3115/1289189.1289191.

[5] Bird, Steven, Loper E., and Klein E., "Natural Language Processing with Python," O'Reilly Media Inc., 2009. http://www.nltk.org.

[6] Dror J., Shaharabani D., Rafi Talmon R., and Wintner S., "Morphological Analysis of the Qur'an," *Literary and Linguistic Computing*, vol. 19, no. 4, pp. 431-452, 2004.

[7] Dukes, K., "Quranic Arabic Corpus," School of Computing, University of Leeds, UK, 2011. http://corpus.quran.com.

[8] Dukes K., Atwell E., Habash N., "Supervised Collaboration for Syntactic Annotation of Quranic Arabic," *The Language Resources and Evaluation*

Conference (LREC), Valletta, Malta, March, 2011.

[9] Dukes K. and Buckwalter T., "A Dependency Treebank of the Quran using Traditional Arabic Grammar," *The 7th international conference on Informatics and Systems (INFOS 2010)*, Cairo, Egypt, 2010.

[10] Dukes K. and Habash N., "Morphological Annotation of Quranic Arabic," *The Language Resources and Evaluation Conference (LREC)*, Malta, 2010.

[11] Frost R. A., and Hafiz, R., "A New Top-Down Parsing Algorithm to Accommodate Ambiguity and Left Recursion in Polynomial Time," *SIGPLAN Notices*, vol. 42, no. 5, pp. 46-54, 2006.

[12] Gabbard R. and Kulick S., "Construct State Modification in the Arabic Treebank," *Association for Computational Linguistics* (*ACL-08): HLT, Short Papers*, pages 209–212, Columbus, Ohio, June 2008. DOI: 10.3115/1557690.1557750.

[13] Green S., Sathi C., and Manning C. D., "NP Subject Detection in Verb Initial Arabic Clauses," *The Third Workshop on Computational Approaches to Arabic Script-based Languages (CAASL3)*, 2009.

[14] Habash N., "Introduction to Arabic Natural Language Processing," Morgan & Claypool Publishers, 2010.

[15] Habash N., Faraj R., and Roth R., "Syntactic Annotation in the Columbia Arabic Treebank," *MEDAR International Conference on Arabic Language Resources and Tools*, Cairo, Egypt, 2009.

[16] Habash N. and Roth R., "CATiB: The Columbia Arabic Treebank". The Association for Computational Linguistics (ACL-IJCNLP) 2009 Conference Short Papers, pp. 221–224, Suntec, Singapore, August 2009. DOI: 10.3115/1667583.1667651.

[17] Hajic J., Hladká B., and Pajas P., "The Prague Dependency Treebank: Annotation Structure and Support," *The IRCS Workshop on Linguistic Databases*, pp. 105–114, Philadelphia, University of Pennsylvania, 2001.

[18] Klein D. and Manning C. D., "Accurate Unlexicalized Parsing," *The 41st Meeting of the Association for Computational Linguistics (ACL'03)*, 2003. DOI: 10.3115/1075096.1075150.

[19] Kruijff G., "Dependency Grammar," *The Encyclopedia of Language and Linguistics*, Second edition, Elsevier Publishers, 2006.

[20] Kulick S., Gabbard R., and Marcus M., "Parsing the Arabic Treebank: Analysis and Improvements," *The Treebanks and Linguistic Theories Conference*, pp. 31–42, Prague, Czech Republic, 2006.

[21] Maamouri M., Bies A., Buckwalter T., and Mekki W., "The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus," *NEMLAR International Conference on Arabic Language Resources and Tools*, pp. 102–109, 2004.

[22] Maamouri M., Bies A., Krouna S., Gaddeche F., and Bouziri B., "Penn Arabic Treebank Guidelines," *Linguistic Data Consortium*, 2009.

[23] Marton Y., Habash N., and Rambow O., "Improving Arabic Dependency Parsing with Lexical and Inflectional Morphological Features," *The NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pp. 13–21, Los Angeles, CA, USA, Association for Computational Linguistics, June 2010.

[24] Nivre J., Hall J., Nilsson J., Chanev A., Eryigit G., Kubler S., Marinov S., and Marsi E., "MaltParser: A Language-independent System for Data-driven Dependency Parsing. Natural Language Engineering," vol. 13, no. 2, pp. 95–135, 2007. DOI: 10.1017/S1351324906004505.

[25] Othman E., Shaalan K., and Rafea A., "A Chart Parser for Analyzing Modern Standard Arabic Sentence," *The MT Summit IX Workshop on Machine Translation for Semitic Languages: Issues and Approaches*, pp. 37–44, 2003.

[26] Owens J., "The Foundations of Grammar: An Introduction to Medieval Arabic Grammatical Theory," John Benjamins Publishers, 2008

[27] Qian S., "Lecture Notes in Natural Language Processing Applications," Calligramme, LORIA & INRIA Nancy Grand-Est Research Centre, France, 2010.

[28] Smrž O., Bielický V., Kouˇrilová I., Kráˇcmar J., Hajic J., and Zemánek P., "Prague Arabic Dependency Treebank: A Word on the Million Words," *The Workshop on Arabic and Local Languages (LREC 2008)*, pp. 16–23, Marrakech, Morocco, 2008.

[29] Smrž O. and Hajic J., "The Other Arabic Treebank: Prague Dependencies and Functions," *Ali Farghaly, editor, Arabic Computational Linguistics: Current Implementations*, CSLI Publications, 2006.

[30] Soudi A., Bosch A., and Neumann G., "Introductory Chapter, Arabic Computational Morphology: Knowledge-based and Empirical Methods," Springer (Text, Speech, and Language Technology series, edited by Nancy Ide and Jean V´eronis, volume 38), 2007, ISBN 978-1-4020-6045-8, 2007.

[31] Tounsi L., Attia M., and Genabith J., "Automatic Treebank-Based Acquisition of Arabic LFG Dependency Structures," *The EACL 2009 Workshop on Computational Approaches to Semitic Languages*, pp. 45–52, Athens, Greece, 2009. DOI: 10.3115/1621774.1621783.

Appendix A

Each of the tables shows the grammar rules for a given Sura. The combined grammar results from putting together the three grammars.

Combined Grammar Rules

S_prod1 = CFGProduction(S, [VS, CJ, VS])
S_prod2 = CFGProduction(S, [VS]) #VS: Verbal Sentence
VS_inv_prod = CFGProduction(VS, [VT]) # VT: Verb
VS_inv_prod1 = CFGProduction(VS, [VT, NN, NN, PP])
VS_inv_prod2 = CFGProduction(VS, [VT, PP])
VS_inv_prod3 = CFGProduction(VS, [VT, PP, PP])
VS_inv_prod4 = CFGProduction(VS, [VT, PP, CP]) # CP: Conjunctional Phrase عطف
VS_inv_prod5 = CFGProduction(VS, [VT, VS])
VS_inv_prod6 = CFGProduction(VS, [VT, PP, CP])
VS_inv_prod7 = CFGProduction(VS, [VT, NP])
VS_inv_prod8 = CFGProduction(VS, [CP])
VS_inv_prod9 = CFGProduction(VS, [VT, PP, PP])
VS_inv_prod10 = CFGProduction(VS, [NG, VS]) # NG: Negation Particle أداة نفي
VS_inv_prod11 = CFGProduction(VS, [VT, PP, NP])
VS_inv_prod12 = CFGProduction(VS, [TS, VS]) #TS: Tasweef Particle أداة تسويف
VS_inv_prod13 = CFGProduction(VS, [VT, NS]) #NS: Nominal Sentence
VS_inv_prod14 = CFGProduction(VS, [VI, NS]) #VI: Incomplete Verb فعل ناقص
NS_inv_prod1 = CFGProduction(NS, [Starter, Predicate]) # Starter: مبتدأ, Predicate: خبر
NS_inv_prod2 = CFGProduction(NS, [Starter, Predicate, Comma, Predicate])
NS_inv_prod3 = CFGProduction(NS, [Advanced_Predicate, Starter]) #Advanced_Predicate: خبر مقدم
NS_inv_prod4 = CFGProduction(NS, [Starter, Predicate, Comma, Predicate, Comma, Predicate])
Starter_inv_prod1 = CFGProduction(Starter, [NP])
Predicate_inv_prod1 = CFGProduction(Predicate, [NP])
Predicate_inv_prod2 = CFGProduction(Predicate, [NS])
Predicate_inv_prod5 = CFGProduction(Predicate, [VS])
Predicate_inv_prod6 = CFGProduction(Predicate, [VS, CJ, VS, CJ, VS])
Advanced_Predicate_inv_prod = CFGProduction(Advanced_Predicate, [PP, NP])
CP_inv_prod2 = CFGProduction(CP, [PP, CJ, CP]) # CJ: Conjunction حرف عطف
CP_inv_prod1 = CFGProduction(CP, [PP, CJ, PP])
CP_inv_prod3 = CFGProduction(CP, [NP, CJ, NP])
PP_prod1 = CFGProduction(PP, [P, NP]) # P: Preposition حرف جر , NP: Nominal Phrase
NP_prod1 = CFGProduction(NP, [NN]) # AdP: Additional Phrase شبه جملة اضافة
NP_prod2 = CFGProduction(NP, [NN, CnP]) # CnP: Conditional Phrase شرط
NP_prod3 = CFGProduction(NP, [AdP])
NP_prod4 = CFGProduction(NP, [NN, PP])
NP_prod5 = CFGProduction(NP, [Prn]) # Prn: Pronoun ضمير
NP_prod6 = CFGProduction(NP, [RelP])
NP_prod7 = CFGProduction(NP, [AdP, Comma, AdP, Comma, AdP])
NP_prod8 = CFGProduction(NP, [CP])
CnP_prod1 = CFGProduction(CnP, [CN, VS]) #CN: Conditional Article ادةشرط
AdP_prod1 = CFGProduction(AdP, [NN, NP])
AdP_prod2 = CFGProduction(AdP, [NN, AdP])
AdP_prod3 = CFGProduction(AdP, [NN, RelP]) # RelP: Relative Phrase صلة
RelP_prod = CFGProduction(RelP, [Rel, VS]) #Rel: Relative Noun اسم موصول

G1 (Al Nas)
S → VS VS → VT, VS VS → VT, PP, PP PP → P, NP NP → AdP, Comma, AdP, Comma, AdP NP → AdP NP → NN AdP → NN, NP AdP → NN, RelP RelP → Rel, VS
10 rules

G2 (Al Falaq)
S → VS VS → VT, VS VS → VT, PP, CP VS → VT PP → P, NP NP → AdP NP → NN NP → NN, CnP NP → NN, PP AdP → NN, NP AdP → NN, RelP RelP → Rel, VS CnP → CN, VS CP → PP, CJ, CP CP → PP, CJ, PP
15 rules

G3 (Al Ikhlas)
S → VS VS → VT VS → VT, NS VS → NG, VS VS → VI, NS NS → Starter, Predicate NS → Starter, Predicate, Comma, Predicate, Comma, Predicate NS → Advanced_Predicate, Starter Starter → NP Predicate → NP Predicate → NS Advanced_Predicate → PP, NP NP → Prn NP → NN
14 rules