1-1-2015

Arabic pro-drop

Mansour Ibrahim Altamimi
mansour.tamimi@gmail.com

Arabic Pro-Drop

by

Mansour Ibrahim Altamimi

Thesis

Submitted to the Department of English Language and Literature

Eastern Michigan University

in partial fulfillment of the requirements

for the degree of

MASTER OF ARTS

in

English Linguistics

Thesis Committee:

T. Daniel Seely, Ph.D, Chair

Damir Cavar, Ph.D.

Beverley Goodman, Ph.D.

January 1, 2015

Dedication

For my parents Norah and Ibrahim Altamimi, and my brothers and sisters.

Acknowledgments

I would like to take the opportunity to express gratitude to several individuals who played monumental roles in encouraging and supporting me during the completion of this thesis. First, I am obliged to express my sincerest thanks to Professor T. Daniel Seely, who served as my advisor and thesis chair. Before entering the linguistics program at Eastern Michigan University, I heard enthusiastic feedback about his approach to linguistics and his upbeat attitude, which motivated me to take his classes. After I had the opportunity to meet him, the initial positive impression I gathered was confirmed. Each of his classes in which I participated had a significant impact on my intellectual development, and led to my decision to conduct my research in syntax. While he was supportive, encouraging, and friendly, he did not simply accept what I submitted, but rather challenged my thought process and instincts. He stimulated my thinking in connection with every theory I encountered, and in particular made me comfortable enough to engage in discussion without fear of criticism. I will miss our frequent casual conversations, during which we discussed cultural differences and similarities.

Second, Dr. Damir Cavar, a committee member and my instructor for chapters four and five of my thesis, warrants special mention. His classes, Statistics for Language Study and Technology for Language Documentation, were particularly inspiring and helped me focus on the subject matter of my thesis. Despite the fact that he took a position at another institution before the completion of my thesis, and thus did not have any professional obligation to continue counseling me, we maintained our connection, just as if he had never left. He found a way to communicate with me so that the concepts conveyed were relatively easy to understand, but at the same time thought-provoking. He provided many ideas and suggestions during the time I worked on my thesis.

Third, committee member Dr. Beverley Goodman stands out in my mind and heart as the "soul" of the Linguistics Department. Her instruction during her Phonological Analysis classes better prepared me to incorporate transcriptions of my mother language (as well as my dialect) in an academically and phonologically proper manner. Dr. Goodman's respectful nature made it easy to engage her, and I thank her for taking time to read through my thesis. She never made it seem like an imposition, and devoted far more of her time than I could have hoped.

I also would like to extend my gratitude to the Linguistic Data Consortium (LDC), which allowed me to examine the OntoNotes Arabic treebank, and particularly to Ann Bies and Ilya Ahtaridis. The opportunity to make use of the treebank was essential to the success of this thesis.

Finally, I would be amiss if I did not also thank my family for their continuing support during my master's career in particular, but also through my entire development and evolution. Their help, as well as that of my mentors mentioned above and many of my peers, has instilled certain values, which have kept me curious, courageous, and motivated through this entire experience.

Abstract

The goal of this thesis is to analyze pro-drop in Arabic in terms of its syntactic properties and the nature of its optionality in discourse. The paper begins with a comprehensive review of five syntactic properties assumed to be universal to pro-drop languages, and discusses the status of each in both Standard Arabic and the Saudi Najdi dialect—the latter having seen little previous research in this area. The conclusion is that Najdi is very nearly syntactically identical to Standard Arabic with regard to the null subject parameter. The topic then shifts to discourse, and what factors may trigger pro-drop usage. This topic is investigated by utilizing the Python programming language to analyze a corpus. New data suggest that there is little variation in pro-drop with regard to the type of nearby verb, when classifying verbs as psych or action. Other data suggest that wh-phrases have a significant impact on the occurrence of pro-drop.

Table of Contents

List of Tables

Chapter 1: Introduction

1.1 Background

Arabic is a Semitic language spoken natively by approximately 300 million people, located primarily throughout the Middle East, North Africa, and the Horn of Africa. Arabic covers a wide array of dialects and languages, not all mutually intelligible, descended from Classical Arabic. Modern Standard Arabic (hereinafter "Standard Arabic") is the formal, standardized, literary form of Arabic, which is taught in schools, used in most written media, and spoken in contexts such as speechmaking and broadcasting. Colloquial Arabic encompasses many dialects that show variation in phonetics and syntax.

Arabic is one of a category of languages known as null-subject languages. In these languages, it is grammatically correct to omit subjects (thus "null"). Being a null-subject language, Arabic exhibits a phenomenon known as pro-drop, where subject pronouns are omitted yet semantic information remains, so that the meaning of a sentence can be determined from the grammatical context.

A number of syntactic properties seem to correlate with whether a language is null-subject or non-null-subject. In null-subject languages, the set of properties related to subject dropping is associated with the null-subject parameter. Each null-subject language—or even dialects of a null-subject language—can be said to have its own settings for the null-subject parameter, which may differ from that of other languages. See Chapter 2 for an in-depth discussion of pro-drop, and five common null-subject parameter properties.

1.2 Statement of Problem

There are three problems addressed in this work. First, although the syntactic features of Standard Arabic pro-drop have been studied previously, there is contention in the literature about

some of the properties, and few or no thorough examinations exist of all five core null-subject parameters with regard to the language. Second, the Saudi Najdi dialect of Arabic has heretofore received little attention by linguists. The null-subject parameter has not been examined in this dialect as it has for Standard Arabic and other dialects. Finally, previous studies of pro-drop in Arabic have primarily focused on its syntactic features, and it has been infrequently discussed with an emphasis on discourse. For instance, there is little available information on what aspects of discourse affect the optionality of Arabic pro-drop.

1.3 Purpose of Study

This study has three primary goals. The first is to thoroughly explore five properties often considered common to null-subject languages. The second is to use the aforementioned exploration of the null-subject parameter to form conclusions about the null-subject parameters of both Standard Arabic and Saudi Najdi Arabic. The discussion of Standard Arabic will be largely a literature review, with an emphasis on clarification of misconceptions about properties, which have been controversial among linguists in the past. Conversely, the investigation of Najdi will be a new addition to the literature. The null-subject parameter of Najdi will be compared and contrasted with that of SA. The third goal is to investigate the occurrence of Arabic pro-drop from a discourse perspective, and present new data on the optionality of pro-drop in Standard Arabic.

1.4 Data Corpus

The data corpus used in chapters 4 and 5 is the OntoNotes Release 5.0 (Linguistic Data Consortium catalog number LDC2013T19) Arabic corpus (cf. Weischedel, 2013). This corpus, originally released in 2010 as “Arabic Treebank: Part 3 v3.2” (LDC2010T08), consists of 339,722 Modern Standard Arabic words sourced from 599 newswire stories from the Lebanese

An Nahar News Agency. The stories date from the 15th day of each month ranging from January to December of 2002 (Maamouri et al., 2008).

1.5 Limitations

A limitation of any analysis in this work stemming from the corpus discussed above is the scope of the corpus itself. The data in the corpus is sourced from a single news agency, and the news stories within it span a single year. However, given the consistency of Standard Arabic across the Arabic-speaking world, it seems a fair assumption that, on the whole, valid generalizations from this data to the Standard Arabic of recent years are possible.

1.6 Scope

Research regarding the null-subject parameter focuses on five core properties (defined in Chapter 2): missing subjects, free inversion, violation of the that-trace effect, long wh-movement, and null resumptive pronouns. The discussion of the null-subject parameter in this work is restricted to these properties.

Chapter 3, the chapter on the null-subject parameter of Arabic dialects, features a literature review detailing previous studies of several Arabic dialects; however, the primary concentration of the chapter is the Saudi Najdi dialect. This concentration limits the majority of the chapter's scope to a single dialect, and no universal statements about Arabic dialects are put forth in the chapter.

Chapter 5 is limited in scope to two discourse analyses. It is intended to provide a relatively small set of experimental data about pro-drop in discourse, and is not a full, extensive study.

1.7 Conventions

Examples and illustrations in this thesis are intended to consistently make use of several conventions, which will be outlined in this section. Text examples are comprised of three lines. The top line contains the original language text. The second contains direct English translations of each word on the first line, generally aligned directly underneath the respective word of each. The second line often includes semantic/case/tense information attached to some or all words. The final line is a colloquial English translation as a complete sentence. A leading asterisk (*) on a sentence denotes ungrammaticality. In some cases the final English translation is ungrammatical even though the sentence in the original language is grammatical. An empty category is denoted with an italicized *e*. A dropped subject is denoted by *pro*. Index information is supplied with subscripted numbers or letters.

Abbreviations that frequently appear in this work are AGR (agreement), COMP (complementizer), ECP (Empty Category Principle), INFL (inflection), NP (noun phrase), NSL (null-subject language), NNSL (non-null-subject language), SA (Standard Arabic), SNA (Saudi Najdi Arabic), TNS (tense), and VP (verb phrase). Each is introduced before its first use in the following chapters, but reproduced here for convenience.

Chapter 2: Standard Arabic as a Null-Subject Language

2.1 Introduction

Since at least the 1980s, linguists have explored and attempted to explain syntactic similarities and differences between languages. One manner of categorization involves labeling languages as null-subject or non-null-subject. As these names imply, the category to which a given language belongs hinges on a property of subjects in the language: specifically, it depends on whether subjects can be completely omitted—thus "null"—in particular circumstances. This act may be referred to as "dropping" a subject, or permitting an "empty" subject, commonly known as "pro-drop."

The following two paragraphs serve as a brief introduction to null subjects. A more exhaustive, technical discussion can be found in section 2.2.1 later in this chapter. A simple example of dropping a subject in Spanish, from Rothman and Iverson (2007), is illustrated in (1)a and (1)b below.

(1)

a. Yo hablo francés.
I speak (1stSg) French
"I speak French."

b. Hablo francés.
Speak (1stSg) French.
"I speak French"

Both Spanish clauses of (1) are grammatical, despite the omission of the subject "Yo" in (1)b. A clause such as that in (1)b is said to have a null subject. In the case of Spanish, the "o" at the end of the verb "hablo" specifies a singular, first-person subject. It will be shown that such information on verbs is important in licensing null subjects in these languages. Languages such as Spanish, whose verbs provide necessary information to identify covert subjects, are referred to

as having “rich agreement.” Researchers have speculated about the significance of the relationship between pro-drop and rich agreement; it has been suggested that there is a strong correlation between languages with rich agreement and languages that permit null subjects. A complete discussion of agreement and its relation to null subjects occurs later in this chapter.

Alternatively, the English equivalents of (1), seen in (2) below, exemplify why English is considered to be a non-null-subject language.

(2)

a. I spoke French.
b. * spoke French.

In contrast to (1)b, (2)a becomes ungrammatical in (2)b when its subject is not realized. The verb “spoke” could agree with the subject “I,” but also with any subject that could be replaced with the pronouns “we,” “he,” “they,” etc. This is because English does not feature rich agreement. In most cases, English and similar languages lack the context created by rich agreement to permit non-explicit subjects. (Limited circumstances in which a non-null-subject language may make use of empty subjects are discussed later in this chapter.)

An important aspect of the linguistic study of null-subject languages is determining what properties seem to be common to these languages, and infrequent or nonexistent in non-null-subject languages. These properties, one could argue, may be seen as criteria for a true null-subject language. Linguists have identified five core properties that appear to be common to null-subject languages. These five properties are referred to as comprising part of the *null-subject parameter* in these languages, and will be discussed in detail in this chapter.

The primary purpose of this chapter is to discuss the null-subject parameter of Standard Arabic (hereinafter SA). Dialects of Arabic will be examined in Chapter 3. In this chapter, each core NSL property will be discussed in general in the form of a literature review, then analyzed

with regard to Standard Arabic. A conclusion will then be drawn about whether or not SA can be considered a null-subject language based on the analysis of each property.

2.2 Properties of Null-Subject Languages in Standard Arabic

Linguists have identified five basic properties that seem to correlate with null-subject languages. These properties have been studied extensively by Rizzi (1980, 1982), Nicolis (2008), and Camacho (2013) among others. In particular, Berjaoui (2009) included an extensive comparison between the properties of SA and English.[1] The basic finding is that languages that allow pro subjects typically feature these five syntactic properties. The expectation is that SA will also exhibit these five properties. This section will define and discuss each of the five core properties of null-subject languages, discuss each with regard to SA, then form a conclusion for each property regarding the degree to which SA corresponds to other null-subject languages. These conclusions will facilitate the central goal of this chapter, which is to determine the status of the null-subject parameter of SA.

2.2.1 Missing subjects. The first property of null-subject languages to be discussed is missing subjects. They have been investigated and studied by Camacho (2013), Ennaji (1991), Chomsky (1981), Rizzi (1982), and Berjaoui (2009), among others. Each aforementioned work agrees that permitting phonologically null subjects in the subject position of tensed clauses is necessary for a language to be considered a null-subject language.

[1] What makes this work unique from Berjaoui's work is the scope, the methodology, and the hypothesis. This work examines NSL and NNSL in general with respect to the properties of NSLs, specifically comparing the SNA with SA, and investigating the effects of discourse on Arabic pro-drop utilizing Python to analyze an Arabic corpus.

Chomsky (1982), as discussed in Berjaoui (2009), argued that inflection (INFL) regulates the availability of null subjects in many languages. For a sentence with an empty category[2] in the subject position to be grammatical, the empty category must be identified by and co-indexed with INFL; it can then be properly considered pro. A language of this type may be referred to as richly inflected. If an empty category in the subject position is not governed by INFL, the Empty Category Principle[3] (ECP) is violated. Languages whose inflection components cannot govern an empty category are referred to as having poor inflection.

In many languages, rich agreement fulfills the task of governing an empty category in the subject position with INFL. Note that an INFL node can contain several elements, but in the case of rich agreement languages, the agreement (AGR) element alone provides the necessary inflectional features—person, number, and gender—which fully identify an empty subject. Null-subject languages with such rich agreement include Italian, Spanish, Irish, and Chamorro.

The Italian examples in (3) below illustrate pro-drop in a language with rich agreement.

(3)

a. Mang-io le tua torta.
eating-1p-sg your cakes.
"I am eating your cakes."

b. Mang-ia le tua torta.
eating-3p-sg-masc your cakes.
"He is eating your cakes."

[2] Note the distinction between *empty category* and *pro*. An empty category is a phonologically null element, which becomes pro when properly identified.

[3] The ECP requires traces to be properly governed (Chomsky, 1981).

In both (3)a and (3)b, all inflectional features are suffixed to the stem of the verb *mang*. These features provide information that can identify the subject of (3)a as being first person singular, and the subject of (3)b as third person masculine. Thus, the sentences are grammatical despite the empty categories (pro) in the subject positions.

Chamorro offers an interesting case in which pro-drop is conditioned by the features a verb carries. In this language, if a verb displays person and number agreement, a null-subject is required because the use of an overt subject will render a sentence ungrammatical, as seen in (4)a. However, if a verb agrees in only one feature, e.g., in person but not in number, the subject can optionally be overt or null, as shown in (4)b.

(4)

a. Ha-fahan *pro*/*gui' I lepblu.
AGR-buy he the book
"He bought the book."

b. Tai-mamalao (hao)!
AGR.not have-shame you.
"You're shameless."
(from Chung, 1998 via Camacho, 2013)

In (4)a the pronoun *gui'*, being overt, results in the sentence being ungrammatical because the verb agrees with the subject in both person and number. In (4)b, the pronoun *hao* can correctly be null or explicit because the verb agrees with the subject in number but not in person.

Languages with poor inflection, as opposed to rich or no inflection, are generally non-null-subject languages. Poor inflection means verbs in these languages cannot provide all of the inflectional features (person, gender, and number) and thus, by hypothesis, an empty subject cannot be properly identified or governed. Examples of such languages are German and English. Consider the English examples in (5).

(5)

a. * Have been writing a novel.
b. * Watched a movie.
c. * Will look it up.
d. * Arrived.

Each sentence of (5) has an empty category in the subject position, and each is ungrammatical. To account for such cases, Chomsky (1981) proposes that in each sentence, INFL fails to identify the empty subject. The result is a violation of the ECP. The INFL features that a verb in a NNSL picks up do not have the power to license pro.

It is important at this point to address the fact that rich agreement is not the sole means by which a language can allow missing subjects. Despite the presence of the INFL and AGR nodes in languages such as Chinese, Japanese, and Indonesian, these languages show no signs of verb-subject agreement. INFL cannot be the licenser of pro-drop in these languages, yet these languages exhibit pro-drop. Figures (6) and (7) below illustrate Chinese and Japanese examples that contain both null pronouns and no agreement features on verbs.

(6)

Ø kanjian ta le.
see he LE
'pro saw him.'
(Chinese, from Neeleman & Szendrői, 2007; Huang, 1998: pp. 533, 563)

(7)

Ø sike- ni otita
exam-DAT failed
"pro failed the exam."
(Japanese, from Neeleman & Szendrői, 2007; Huang, 1998: pp. 533, 563)

The type of pro-drops seen in these examples, in which there are no overt morphological clues of inflectional features yet the pro remains grammatical, are referred to as radical or discourse pro-drops (Neeleman & Szendrői, 2005). It is important to note that the null subjects of

(6) and (7) cannot be explicitly identified without more context from the discourse; radical pro-drop differs significantly from INFL-governed pro-drop in these cases.

Neeleman and Szendrői (2005) claim that languages must be of a specific pronomial paradigm to allow radical pro-drop. Specifically, their central claim is that "a language will only allow radical pro drop if its personal pronouns are agglutinating for case, number, or some other nominal feature." Japanese pronouns are of an agglutinating case morphology.[4] This is illustrated in (8).

(8)

Kare-ga kare-o setokusuru
he-nom he-acc persuade
"He persuades him."
(Japanese, cf. Neeleman & Szendrői, 2005)

Contrast the agglutinative case morphology in (8) with, for example, the English *he* and *him*, or the Dutch *hij* and *hem*, both examples of fusional case morphology. Chinese pronouns differ in morphology from Japanese. They are invariant in subject and object positions, and agglutinate not for case, but for number. This is illustrated in (9).

(9)

a. wǒ-men-Ø
I-pl-case
"we/us"

b. wǒ-de
I-poss
"my/mine"
(Chinese, cf. Neeleman & Szendrői, 2005)

[4] According to Nellman et al. an agglutinating case morphology features identifiable affixes for case or agreement.

It is possible for a language to have Chinese-like invariant pronouns but disallow pro-drop. As an example, Neeleman and Szendrői (2005) cite Jamaican Creole. They argue that the reason it disallows pro-drop is its pronouns are invariant for *all* nominal features (cf. Neeleman and Szendrői, 2005). As the primary concentration of this work is Arabic—for which comparison with radical pro-drop languages has little relevance—languages featuring radical pro-drop will not be discussed beyond the brief introductory material in this section.

This section has expanded upon the introduction of missing/null subjects. A missing subject in regard to the null-subject parameter is defined as a phonologically unrealized subject in the subject position of tensed clauses. The chapter discussed three major types of languages that treat missing subjects differently. Two types of languages may permit null subjects: those with rich agreement, where the pro is governed by AGR/TNS under INFL, and those with no agreement but with an agglutinative pronoun case morphology. The third type—languages with poor inflection (i.e., NNSLs)—do not permit missing subjects.

2.2.2 Missing subjects in Standard Arabic. Now that the general background of missing subjects has been presented from the literature, missing subjects in SA specifically can be discussed. Simply put, SA permits missing subjects in the subject position of tensed clauses. The following examples demonstrate missing subjects in SA.

(10)

a. qara-ʔa kita:ban
read-3p.sg.masc a book
"He read a book."

b. katab-at riwayetan
wrote-3p.sg.f a novel
"She wrote a novel."

c. wasˁal-u: lbariħa

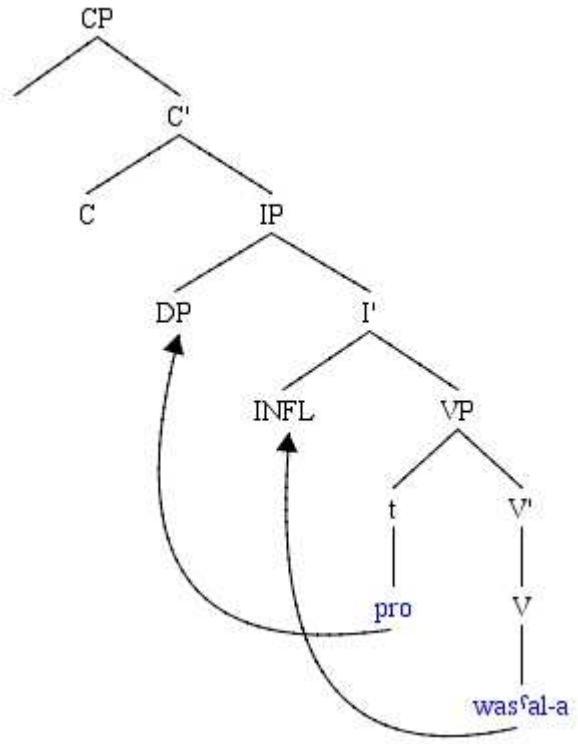
arrived-3p.pl last night
"They arrived last night."

d. wasʿal-a.
arrived-3p.sg.m-nom
"He arrived."

Each SA sentence above lacks an overt subject in the subject position, and each sentence is grammatical. SA is a rich agreement language as discussed in the previous section. AGR in SA can provide the person, gender, and number features necessary to identify an empty subject (Berjaoui, 2009; Alotaibi, 2013; AlAlamat, 2014). In SA, as in Italian and Spanish, AGR features are suffixed to verbs. This can be seen in the examples in (10) above.

The tree below in (11), a representation of the sentence in (10)d, illustrates rich agreement in SA. In (11), the subject position is occupied by *pro*, which is co-indexed with the AGR marker [a] suffixed to the verb *wasʿal*. In the standard derivation of this sentence, *pro* starts in the specifier of VP as shown in the tree. It will get a theta role from the verb, then the verb will rise to INFL to instantiate the SA word order SVO (Berjaoui, 2009; Alotaiby, 2013).

(11)



An interesting feature of SA syntax is the clitic. Clitics are explained by Alotaiby et al. (2012) as "affixes realizing an otherwise unexpressed argument." In SA, suffixes standing for direct or indirect objects can be attached to verbs, which may or may not be accompanied by a free pronoun in most cases (cf. Hahn, n.d.). These suffixes are clitics. An example of an object clitic is seen in (12) below.

(12)

ra'aytu-ka
saw.1sg-you.ACC
"I saw you."

In (12), the clitic is *ka*, standing for the object. It is important to note the distinction between *tu*, which is an agreement marker, and the clitic.

Clitics, by their nature being suffixed to verbs and standing in for a covert (object) pronoun, are similar to the agreement markers on SA verbs that identify subjects. Due to this

similarity, clitics can be a source of confusion when investigating agreement and missing subjects in SA. Some linguists, for example Berjaoui (2009), make no distinction between the clitic and agreement. This work will adopt the assumption that there are important distinctions between clitics and agreement in SA. For one, object clitics can attach to different hosts beyond verbs, such as prepositions and nouns (cf. Jlassi, n.d.). An example of a clitic with a noun host is seen in (13) below. Conversely, subject agreement markers can only appear on verbs.

(13)

baytu-ka
house-you.GEN
"Your house."

The overall point of the discussion about the distinction between clitics and agreement in SA is that missing subjects in SA, just as in other NSLs such as Spanish and Italian, are licensed by AGR. The author believes that the conclusion advanced by Berjaoui (2009)—that "--pro-- in the subject position is properly governed by the Clitic with which it is coindexed"—is based on a misinterpretation of clitics in SA. This work assumes that the licenser of pro in SA is AGR under INFL. Furthermore, Berjaoui's claim that "the occurrence of Clitics in SA allows [the Missing Subject property] in the language" is incorrect, because it is based on the same misconception.

It can be concluded that SA permits missing subjects in much the same way as other rich agreement languages. In regard to this first feature of the null-subject parameter, SA strongly parallels other rich agreement null-subject languages. This is the first step toward showing that SA exhibits all of the core NSL properties. Next, the second property of the null-subject parameter, free inversion, will be investigated.

2.2.3 Free inversion. The second core property suggested as common to null-subject languages is free inversion. According to pioneers in linguistics such as Kayne (1975), Chomsky

(1981), Rizzi (1982), Safir (1985), Camacho (2013), and Cognola (2013), the availability of free inversion in a language is an indication that the language is a null-subject language. Free inversion languages allow the subject to appear on either side of the verb in any sentence. Put differently, a language with free inversion has available an alternative verb-subject (VS) word order, in which the subject is adjoined to the VP, in addition to the default subject-verb (SV) order (Camacho, 2013). The inversion is referred to as "free" because it is not influenced by any trigger; i.e., the movement is optional and not to satisfy some restriction (cf. Riemsdijk & Williams, 1986). Note that free inversion involves only the subject and verb, and not other verb arguments such as the direct object.

Examples (14) and (15) show free inversion in Spanish and Italian, respectively.

(14)

a. Juan contesto la pregunta
Juan answered the question.
"Juan answered the question."

b. *e* Constesto la pregunta Juan.
answered the question Juan
"Juan answered the question."

(15)

a. Le brigate rosse hano telefonato
the brigades red have called
"The red brigades have called."

b.. *e* hanno telefonato le brigate rosse
have called the brigades red
"The red brigades have called."

The sentences in (14) and (15) are all grammatical despite the inversions of word order. Also note the empty categories in (14)b and (15)b. Rizzi (1982), Kayne (1975), and Chomsky (1981) argued that free inversion is associated with VP-adjunction. For this reason an empty

category is left in the subject position, while the NP is adjoined to the VP. Contrast the NSL examples above with that of English, an NNSL which does not feature free inversion, in (16) below.

(16)

a. John left.

b. * Left John.

As stated previously, free inversion is not permitted in NNSLs. For instance, referring to the English example in (16)a, if the subject "John" is moved to the right periphery of the VP, as seen in example (16)b, the sentence will be ungrammatical.

A standard traditional claim, put forth by Chomsky (1981), is that correct formulation of the null-subject parameter requires free inversion. That is, he argues that there is a link between pro subjects and free inversion: if a language allows pro subjects, then that language also allows free inversion. Free inversion is seen as so essential a property because it is thought to be an important reason for the absence of the that-trace effect (the next property discussed in this chapter) in pro-drop languages. However, later work argued that languages exist which allow free inversion but not missing subjects, as well as languages with missing subjects that disallow free inversion (Safir, 1985).

2.2.4 Free inversion in Standard Arabic. Free inversion in SA is illustrated in (17). The SV and VS word orders, in (17)a and (17)b respectively, are both grammatical, and the meaning of the sentence is identical despite the change in word order.

(17)

a. ahmed fataħa el-babə.
Ahmed opened the-door
"Ahmed opened the door."

b. fataħa ahemed el-babə
opened Ahmed el-babə
"Ahmed opened the door."

SA is more complicated because the standard word order is VS, but the standard argument about SA is not that the subject has moved to the end, but that the verb has raised up. So we have a very interesting question here. Is this verb raising or subject inversion? According to Berjaoui (2009),[5] example (18)b is a free inversion case. This work also adopts this assumption.

(18)

a. ahmed fataħa el-babə.
Ahmed opened the-door
"Ahmed opened the door."

b. Fataħ-a el-babə Ahmed.
opened3p.sg.m the door Ahmed
"Ahmed opened the door."

However, when the inversion is shown to be from SVO to VOS as shown in example (18)b, it cannot be said that the verb has raised up, because that would not give the surface structure of VOS. This is the more compelling argument for an inversion in SA because it is the only way to get the order VOS.

The validity (or potential invalidity in other languages) of the additional orders in (18) have no impact on SA's status of allowing free inversion. The only necessary criterion for a language to allow free inversion is that a subject can appear on either side of the verb. In other

[5] For extensive discussion of whether this is a free inversion case or head movement, see Berjaoui (2009), pp. 66–77.

words, the subject NP can be moved to the right periphery of VP. This data shows that SA correlates with other NSLs in its use of free inversion.

2.2.5 Violation of that-trace effect. The phenomenon known as the that-trace effect has been investigated extensively (Chomsky, 1981; Berjaoui, 2009; Rizzi, 1982; Camacho, 2013; Bayer & Salzman, 2013). Prior to discussing the *violation of* the that-trace filter, which is a feature common to NSLs, it is necessary to review the meaning of the that-trace effect. The following section is dedicated to that purpose.

2.2.5.1 That-trace effect. Languages in which an overt complementizer, such as the English *that,* cannot be followed by a subject trace are said to exhibit the that-trace effect. In other words, in these languages a subject cannot be extracted when it follows *that* (Chomsky, 1986a; Kayne, 1984; Perlmutter, 1971). Languages that exhibit the that-trace effect are said to obey the that-trace filter. Typically, these languages are NNSLs. The examples in (19) and (20) illustrate the that-trace effect in English and Finnish, respectively.

(19)

a. What$_i$ do you think [$_{CP}$ **that** [$_{IP}$ **he** read *t* last night?]]
b. *Who$_i$ do you think [$_{CP}$ **that** [$_{IP}$ ***t*** read last night?]]
(English)

(20)

a. Kenet$_i$ sinä luulet [$_{CP}$ t' **että** [$_{IP}$ **Pekka** näki t_i]]
who you think that Pekka saw
"Who do you think that Pekka saw?"

b. *Kenet$_i$ sinä luulet [$_{CP}$ t' **että** [$_{IP}$ **t_i** näki Pekka]]
who you think that saw Pekka
*"Who do you think that saw Pekka?"
(Finnish)

The sentences in (19)a and (20)a are grammatical because the complementizers *that/etta* are followed by an overt subject in each case. Conversely, (19)b and (20)b are ungrammatical because in both cases a subject wh-trace follows *that/etta*.[6]

2.2.5.2 Violation of that-trace filter in null-subject languages. Languages that permit violation of the that-trace filter are of interest to linguists because this property seems to cluster with the other null-subject parameter properties discussed in this chapter. It has been suggested by Chomsky (1981), Berjaoui (2009), Rizzi (1982), and Camacho (2013), among others, that permitting violation of the that-trace filter is a property common to NSLs.

Violation of the that-trace filter in Italian and Spanish (both NSLs) is illustrated in (21).

(21)

a. chi_i credi [CP t' **che** [IP **t_i** [vp comprera questi]]]
who think-you that will-buy these paintings.
*"Who do you think that will buy the paintings?"
(Italian)

b. ¿$Quién_i$ piensa-s [CP t' **que** [IP **t_i** [vp vendrá]]]
who think-you that will come.
*"Who do think that will come?"
(Spanish)

The English translations of (21) are ungrammatical, as it has been previously shown that English exhibits the that-trace effect. However, the Italian and Spanish sentences are grammatical despite both featuring traces immediately following the complementizers *che* and *que* when the wh-phrases are extracted.

[6] Note that "follows" here does not imply a strict linear order; rather, it refers to a structural relationship.

There are multiple hypotheses about why some languages are able to violate the that-trace filter. Chomsky (1981) and Rizzi (1982) reasoned that movement in these languages originates from the post-verbal position, and therefore these languages only *seem* to violate the that-trace effect, while in languages such as English the movement originates from the spec of IP. It may also be argued that in these languages, AGR under INFL can properly command the trace of a subject immediately following *that*, and therefore the ECP is not violated.

In summary, it has been posited that in NSLs, sentences violating the that-trace filter will be grammatical, in stark contrast to NNSLs, which are expected to obey the filter. This means that in NSLs, it is grammatical to extract the wh-phrase over the complemetizer *that.*

2.2.6 That-trace filter in Standard Arabic. Thus far, the null subject parameter of SA has exhibited every expected property. The expectation, then, is that SA will again parallel other NSLs and permit violation of the that-trace filter. Consider example (22) below.

(22)

*man$_i$ taʔtaqid-u [$_{CP}$ ʔanna [$_{IP}$ *t* [fataħa l-bab-a]]]
who think-2ndSgMasc that opened the-door-Acc
*"Who do you think that opened the door?"

Surprisingly, the sentence in (22), in which the wh-phrase of the lower clause is extracted over the complementizer *ʔanna*, is ungrammatical. SA seems to obey the that-trace filter here, which suggests it patterns with NNSLs.

The seeming patterning of SA with NNSLs with regard to the that-trace effect has puzzled linguists and has led to considerable contention in the literature (see Berjaoui, 2009 and Aoun, 1981). This work purports that SA, in contrast to what (22) seems to suggest, patterns more closely with NSLs with regard to the that-trace filter than NNSLs. The following paragraphs will explain this view.

First, it has been argued in the literature by several linguists (cf. Berjaoui, 2009; Akkal, 1996; Ouhalla, 1997) that the subject-verb-object order found in Arabic is more accurately topic-verb-object. Berjaoui's analysis of this property concluded that SA simply "cannot be analyzed" with regard to the that-trace filter, because the filter assumes that what follows the complementizer should be a subject and not a topic as seen in SA. This work takes a different approach, and shows that even if the that-trace filter is assumed to apply to SA, the language can still be said to violate it as expected; however, this violation occurs in a slightly different manner than in other NSLs. An alternative analysis of the ungrammaticality of example (22) is that the SA complementizer has a unique characteristic that distinguishes SA from other languages—namely NNSL and NSL—with respect to the that-trace effect. Consider example (23).

(23)

a. ʔa-ðunn-u [$_{CP}$ **ʔanna** [**ali-an** [$_{VP}$ wasˤal-a]]].
think-I that Ali.ACC arrived
"I think that Ali arrived.

b. ʔa-ðunn-u [$_{CP}$ **ʔanna**-hu [$_{VP}$ wasˤal-a]]].
think-I that-he arrived
"I think that he arrived."

c. *ʔa-ðunn-u [$_{CP}$ **ʔanna** [IP ***pro*** [$_{VP}$ wasˤal-a.]]]
think-I that *ec* arrived
*"I think that arrived."

Example (23) shows that *ʔanna* must be followed by an overt accusative element NP. This overt NP could appear as a clitic pronoun attached to *ʔanna*, as seen in (23)b. However, when *ʔanna* is immediately followed by *pro*, the sentence will be ungrammatical as shown in example (23)c. Because these sentences do not include traces from wh-extractions, the assumption is that this phenomenon has nothing to do with the that-trace effect.

Furthermore, there is an important distinction between SA and English, an NNSL, with regard to the that-trace filter. As described in section 2.2.5.1, on the traditional view of the that-

trace filter, an overt subject NP must follow the complementizer. However, in SA what follows *ʔanna* may be *any* NP, and not only a subject (Aoun, 1981). Consider example (24) below.

(24)

a. ʔa-ðunn-u [$_{CP}$ **ʔanna** [**al-hisˁa:n-a** [$_{VP}$ rɪkɛpa-hu]]].
think-I that the-horse.ACC rode.3psg-he
*"I think that the horse he rode."

b. ʔa-dunn-u [$_{CP}$ **ʔanna** [**al-kitāb-a** [$_{VP}$ karaʔa-hu]]].
think-I that the-book.ACC read.past-he
*"I think that the book he read."

In both (24)a and (24)b, an object follows *ʔanna* yet the sentences are grammatical. This example shows further support that SA and (for example) English cannot be treated alike with regard to the that-trace filter.

If SA does not obey the that-trace filter as seen in NNSLs, what accounts for the ungrammaticality of example (22)? In fact, the explanation arises from an independent property of SA, first exemplified above in (23), which states that the complementizer *ʔanna* must be followed by some overt accusative NP (Alotaibi, 2013; Berjaoui, 2009; and Aoun 1981). This principle is characterized in (25).

(25)

ʔanna ⟶ NP.ACC

The ungrammaticality of (22) and (23)c, as well as the grammaticality of the sentences of (24), naturally follow from this principle. In both sentences in (24), the independent principle is upheld because overt accusative object NPs follow *ʔanna*. (23)a and (23)b also satisfy the principle. Finally, the sentences in (22) and (23)c are ungrammatical not because the that-trace filter is obeyed, but because the lack of overt accusative NPs following *ʔanna* violates the

principle of (25). This independently motivated principle appears to adequately explain why SA *seems* to obey the filter in some cases.

In summary, SA also does not pattern completely with other NSLs such as Italian with regard to the that-trace filter: *ʔanna* cannot be followed by a verb as (for example) *che* can in Italian. Importantly, however, *ʔanna* can be followed by an object NP, which cannot happen in languages that obey the that-trace filter. Thus, SA does permit violation of the that-trace filter, and cannot be said to pattern with NNSLs. SA is a very interesting and intriguing example of a NSL which *seems* to obey the that-trace filter due to an independent property of the language. In regard to the expected null subject parameter property, which states that SA should permit violation of the that-trace filter, this work purports that SA can be said to pattern with other NSLs.

2.2.7 Long wh-movement of subjects. Long-wh movement of subjects is typically permitted in null-subject languages, while in non-null-subject languages it is ungrammatical to employ it. Thus, allowing long wh-movement of subjects is often considered a core property of the null subject parameter (Berjaoui, 2009; Camacho, 2013; Rizzi, 1980; Chomsky, 1981). This section will define and discuss long-wh movement as seen in NSLs.

Long wh-movement of subjects is a phenomenon wherein a wh-phrase is extracted from the subject position and moves across more than one bounding node in a single step. The restriction on movement correlates with Chomsky's (1981) subjacency principle, which states that movement must be local. Formally, it states that no rule can relate x, y in the structure

$$\ldots x \ldots [_{\alpha} \ldots [_{\beta} \ldots y \ldots] \ldots] \ldots$$

where α and β are bounding nodes. Bounding nodes are not consistent across languages, but rather are subject to parametric variation. In English, the bounding nodes are considered to be IP/TP and NP.

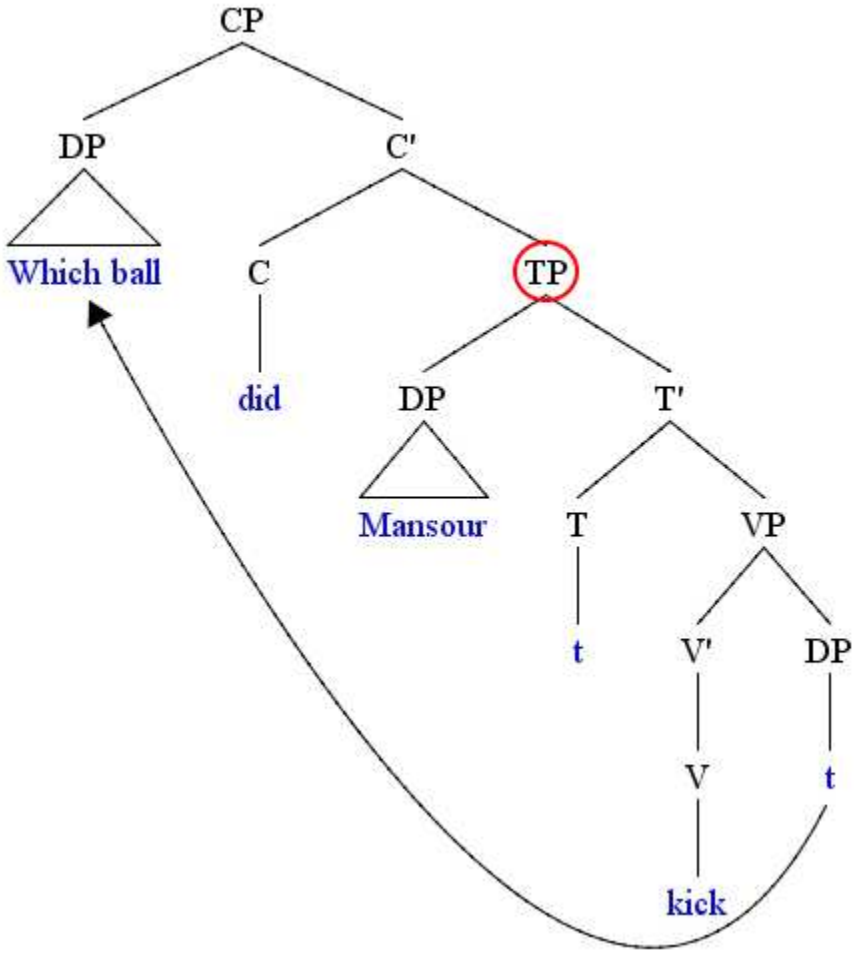
Long-wh movement/subjacency is more easily demonstrated using an NNSL such as English, because NSLs seem to violate the constraint in question. The discussion of long-wh movement will thus begin with English examples, seen in (26).

(26)

a. Which ball did Mansour kick?
b. Which ball did he think Mansour kicked?
c. *Which ball did he believe the claim Mansour kicked?

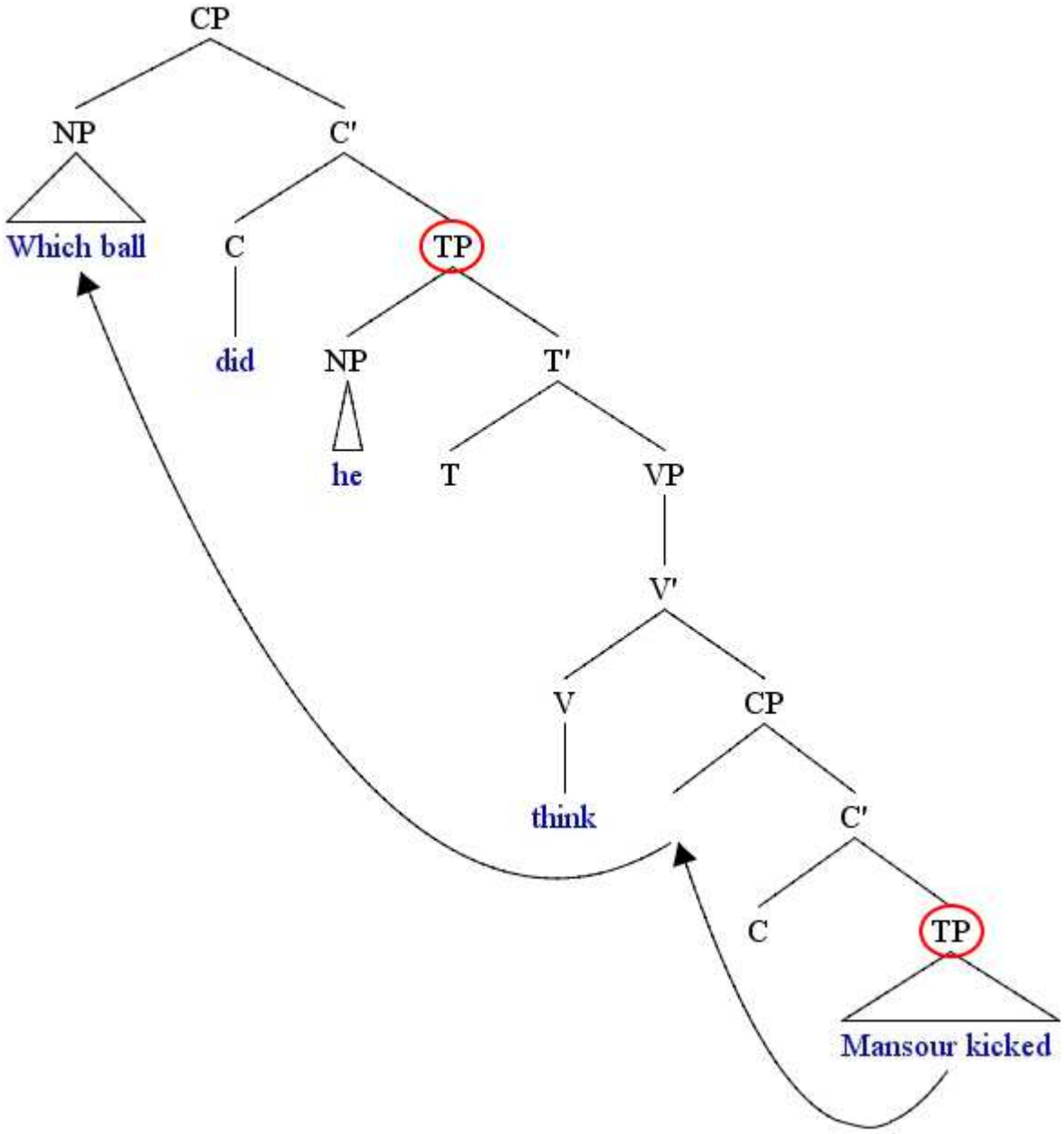
The tree representing (26)a is seen in (27). The sentence is well-formed, because on its way to the front of the wh-question, it crosses only one bounding node, a TP.

(27)



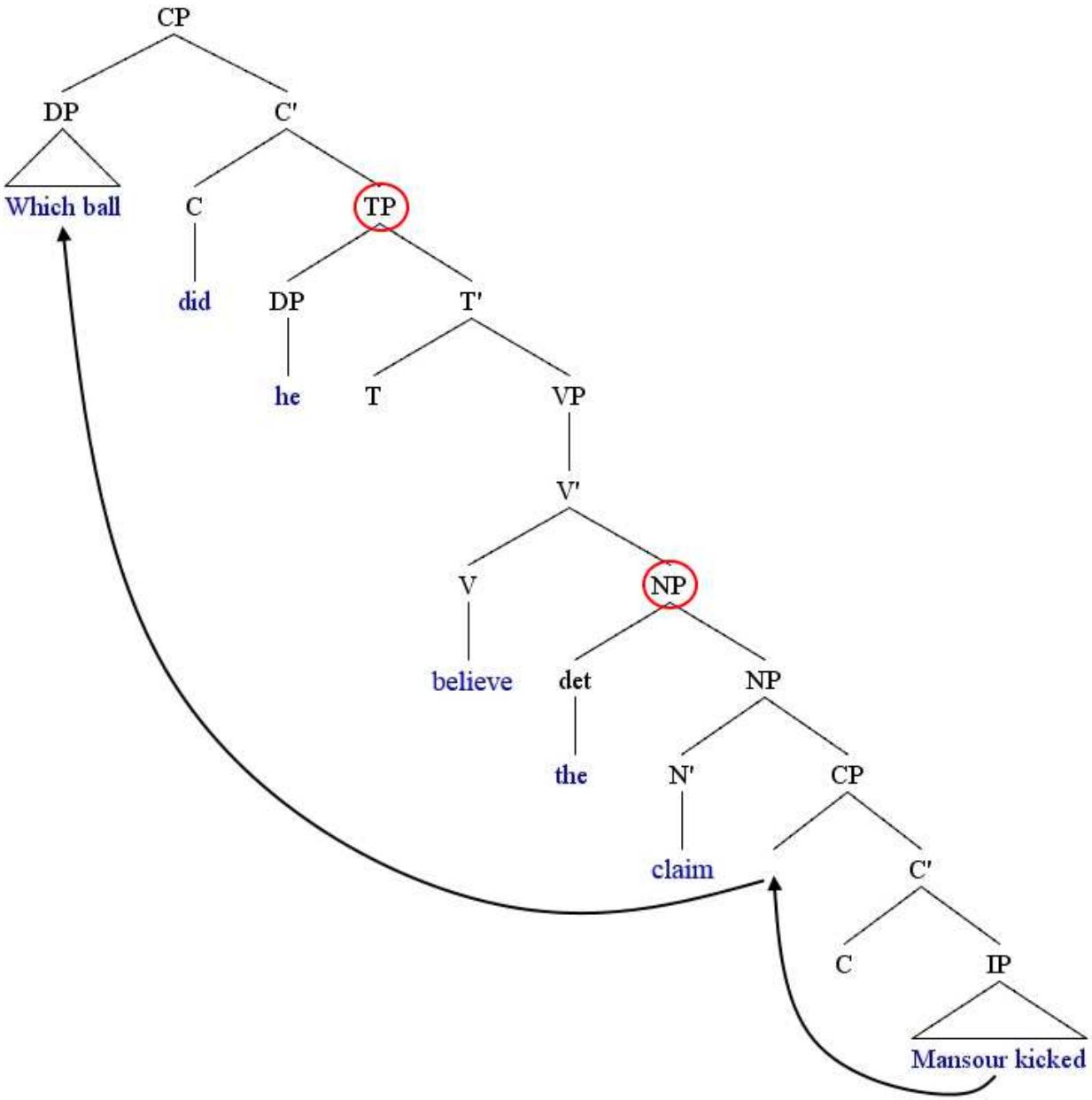
The sentence of (26)b, treed in (27), is also grammatical. Though two bounding nodes must be crossed, the wh-phrase trace is able to first stop at an "escape hatch," the unoccupied spec of CP, and the movement is accomplished in two steps with only one bounding node crossed on each step.

(28)



The sentence of (26)c exemplifies the ungrammaticality of long wh-movement in English. The wh-phrase must cross two bounding nodes in a single step during its extraction, as illustrated in (29). The result is a violation of subjacency and the ECP, which is violated because the trace is not properly governed.

(29)



In contrast to NNSLs, it has been suggested that NSLs exhibit long movement of the subject in wh-questions and matrix clauses. Consider the examples of (30) below from Italian and Spanish.

(30)

a. L'uomo$_i$ [che mi domando [chi *t* abbia visto]]
the-man that CL wonder who has seen

"The man such that I wonder who saw."
(Italian, from Chomsky 1981:240 via Camacho, 2013)

b. ¿Quién$_2$ dijiste (tú) [a quiénes$_3$ (no) les había e$_2$ prestado un montón de dinero e$_3$]?
who said you to whom not CL had lent a lot of money
"Who did you say to whom had (not) lent a lot of money?"
(Spanish, from Camacho, 2013, p.22)

Each example in (30) comes from an NSL and each exhibits long wh-movement of subjects. In example (30)a the wh-word *chi,* which occupies the subject position of the lower clause, crosses more than one bounding node, and yet the sentence is grammatical. The same process is seen in the Spanish sentence.

To summarize, in an NNSL, violations of the subjacency rule and the ECP that arise from long-wh movement of a subject renders any sentence that employs it ungrammatical. In contrast, NSLs typically make free use of long-wh movement of subjects. Thus, its allowance is seen as a core property of NSLs.

2.2.8 Long wh-movement of subjects in Standard Arabic. If SA is to be considered a NSL, it is expected to employ long-wh movement of subjects as other NSLs do, as established in the previous section. Consider the sentences of (31) below, similar in form to the English sentences of (26) in the previous section.

(31)

a. ʔai ħaqibatın mansour akhz-a?
which bag Mansour take-3p.sg.m.past
"Which bag did Mansour take?"

b. ʔai ħaqibatın dann-a mansour akhz-a?
Which bag think-3p.sg.past.m. Mansour take-3p.sg.m.past
"Which bag did he think Mansour took?"

c. *ʔai ħaqibatın dann-a al iddi'ā mansour akhz-a?
which bag think-3p.sg.past.m the claim Mansour take-3p.sg.m.past

*"Which bag did he think the claim Mansour took?"

d. *man taʔtaqid-u man raʔa?
who wonder-2p.sg.m who saw
*"Who do you wonder who saw?"

The examples in (31) demonstrate that SA seems to pattern with NNSLs such as English. The grammaticality of the parallel sentences of (26) and (31) is identical for each sentence. It seems a fair assumption that the bounding nodes of SA are IP and NP, as in English. This conclusion is supported in Berjaoui (2009). This conclusion is unexpected, of course, as SA has patterned with NSLs in the properties discussed previously—meaning that extraction of a wh-word over another wh-word is not permitted in either language. In short, there is no known explanation for exactly why SA breaks pattern with other NSLs with regard to this property. It is a puzzling question worthy of future research. Next, the final NSL property, null resumptive pronouns, will be discussed.

2.2.9 Null resumptive pronouns. Resumptive pronouns are pronouns that appear inside relative clauses and refer to an antecedent in the matrix clause. In the sentence in (32)a, the pronoun *it* coindexed with the NP "a tree" is a resumptive pronoun. In (32)b, the resumptive pronoun is replaced with an empty category, which results in an ungrammatical sentence.

(32)

a. This is a tree$_j$ that when it snows it$_j$ hibernates.
b. *This is a tree$_i$ that when it snows e_i hibernates.

Resumptive pronouns are in complementary distribution with traces. In other words, whenever a trace is possible, a resumptive pronoun is not. This is illustrated by the contrast in the following example:

(33)

a. *The book$_i$ that I read it$_i$.

b. The book that I read $_j$.

The sentence in (33)a is ungrammatical because the resumptive pronoun "it" is in the position of a trace. When it is removed from the sentence, as in (33)b, the sentence becomes grammatical. A resumptive pronoun may thus be necessary when a trace is not available, as seen in (32) above.

The permitting of null resumptive pronouns has been established as a common property of null-subject languages (Berjaoui, 2009; Camacho, 2013; Rizzi, 1982). Sells (1984) defines a null resumptive pronoun as "a pronoun that is interpreted as a bound variable whose antecedent is an operator." Resumptive pronouns occur usually in relative clauses. The following example, originally presented in Berjaoui (2009), is a grammatical Italian sentence with a null resumptive pronoun. In the example, the resumptive pronoun that should play the role of the subject of the verb "has arrived" is null, yet the sentence is grammatical.

(34)

Esta es la mujer [CP que [IP me pregunto [CP quién cree [CP que no [IP *t* venga.]]]

This is the woman who CL wonder who thinks that not come

* "This is the woman who I wonder who believes might come."

(Italian, from Camacho, 2013)

Example (34) demonstrates the valid use of null resumptive pronouns in null-subject languages.

2.2.10 Null resumptive pronouns in Standard Arabic. There is some contention in the literature over whether Arabic permits null resumptive pronouns which, as established in the previous section, is expected if it is to be considered an NSL. Berjaoui (2009) argues that SA disallows null resumptive pronouns. He presented the ungrammatical sentence in (35) and attributes its ungrammaticality to the absence of a resumptive pronoun.

(35)

*el-fatatu llati la ʕaʔlamu man jadunnu ʔanna xaraʒat tuwiffijet
the-girl that not know(1Sg) who thinks that went(3SgFem) died
*”The girl who I don’t know who thinks went out died”.

This sentence is indeed ungrammatical, but its ungrammaticality is not a consequence of the absence of a resumptive pronoun. Rather, it is a result of the complementizer *ʔanna* being followed by a verb. Recall from section 2.4 of this chapter that *ʔanna* must always be followed by an accusative NP.

In contrast to Berjaoui, this work will adopt the assumption that null resumptive pronouns are allowed in Arabic. This assumption is supported by the examples in (36).

(36)

a. [CP [IP haðihi hije eʃ-ʃeʒeratu llati [CP ʔindema [IP janzil-u *e* eθelʒu [IP *e* tusbitu]]]]]
this is the-tree that when fall snow-3p.sg hibernates
*”This is the tree that when it snows, hibernates.” or more correctly
“This is the tree that when it snows, it hibernates.”

b. [CP [IP haʾulāʾi e-rajal el-lazi-na [CP ʔindema [IP yā'ti-a *e* e-ta_ām [IP *e* jākulu:n]]]]]
Those are the-men that-3p.pl.M when come-3p.sg. the-food eat-3p.pl.M
“*Those are the men that when the food comes, eat.”

The SA sentences in (36) are grammatical, and exhibit null resumptive pronouns. It is therefore the conclusion of this work that SA does allow null resumptive pronouns. In this final property, SA correlates with the standard assumption of the null subject parameter of pro-drop languages.

2.3 Optionality of Arabic Pro-Drop

The previous section has shown that pro-drop in Arabic is licensed by agreement. The actual occurrence of pro-drop seems to be fixable in SA. A question that arises is whether there are any syntactic restrictions on dropping a pronoun, or if pro-drop is always syntactically

available to the speaker. “Syntactically available” is used here because the effects of discourse on pro-drop are not discussed at this point, and will be investigated later in Chapters 4 and 5.

The only known syntactic restrictions on Arabic pro-drop occur in *ʔanna* and *ʔan* clauses. Recall from section 2.2.5 that *ʔanna* must be followed by an overt NP. (See example (23).) The mood assigner *ʔan*, in contrast to *ʔanna*, must always be followed by a verb. Example (37) illustrates *ʔan* clauses.

(37)

a. ʔansˁaħu ʔan naðhab ɪla al-madrasatɪ.
suggest (1p.s) that go(1p.pl) to the-school
“I suggest that we go to the school.”

b. *ʔansˁaħu ʔan naħnu naðhaba ɪla al-madrasatɪ.
suggest(1p.s) that we go(1p.pl) to the-school
“I suggest that we go to the school.”

In (37)a, the sentence is grammatical because *ʔan* is immediately followed by the verb *naðhab*, which agrees with the covert *naħnu/we*. When the pronoun is overt, as in (37)b, the sentence becomes ungrammatical because *ʔan* is no longer followed by a verb.

Beyond the aforementioned specific cases of pronouns embedded in *ʔanna* and *ʔan* clauses, pro-drop in Arabic is entirely optional with regard to syntax.

2.4 Conclusion

The goal of this chapter was to investigate the status of the null-subject parameter in SA, and clarify past misconceptions. To accomplish this, it was necessary to address the properties that linguists agree seem to be common to pro-drop languages. Five basic properties have each been defined and discussed in general, then applied to SA.

It has been shown that SA parallels other richly-inflected null-subject languages in its use of missing subjects and free inversion. This work challenged past assumptions that SA does not

exhibit null resumptive pronouns and argued that they are, in fact, permitted. With regard to the third property, violation of the that-trace filter, Arabic is unique relative to other NNSLs. An independent property of SA that the complementizer *ʔanna* must be followed by an overt accusative NP, results in SA *appearing* to obey the that-trace filter. However, the aforementioned independent property, as well as the distinction between Arabic topics and subjects, lends support to this work's assumption that SA's unique treatment of the that-trace filter should be considered a "violation" in terms of the NSL property stating pro-drop languages violate the that-trace effect. Thus, SA again patterns with other NSLs. In regard to the final property, long wh-movement of subjects, the null subject parameter of SA differs from that assumed to be common to NSLs. SA patterns with English, an NNSL, in its disallowance of long wh-movement of subjects. An explanation for this phenomenon does not exist at this time.

Arabic's exhibition of four of five expected properties is an interesting and perplexing result. Exactly how Arabic should be classified, given the current view of the standard null-subject parameter, is not clear. It has vastly more in common with other richly-inflected null-subject languages than any NNSL. Placing it in some third category distinct from other pro-drop languages, to which it shares so many common features, seems incorrect. It is the conclusion of this work that there is a great deal of support for considering SA to be a pure pro-drop language.

Chapter 3: The Null-Subject Parameter of Saudi Najdi Dialect

3.1 Introduction

Chapter 2 established that Standard Arabic can be considered a null-subject language (hereinafter NSL) by discussing its adherence to five core properties of NSLs. The conclusion was that SA exhibited four of the five expected properties. Standard Arabic (hereinafter SA), descended from Classical Arabic, is the formal, literary form of Arabic. There also exist a large number of dialects, also descended from Classical Arabic, used only colloquially. While SA remains a standardized, consistent, formal form of the language across the Arabic-speaking world, there is a great deal of variation in vernacular Arabic dialects. Primarily, dialects have evolved to differ from each other (and SA) phonologically and semantically, but there can be syntactic differences as well. Therefore, it cannot be assumed that each Arabic dialect would adhere to the properties of NSLs in the same manner as would SA. The study of Arabic dialects beyond SA with regard to the null-subject parameter is the focus of this chapter.

Given the huge number and variety of Arabic dialects, as well as the difficulty inherent in classifying and bounding dialects in general, a conclusion about *all* Arabic dialects is not feasible. Instead, the primary focus of this chapter is a single dialect, Saudi Najdi Arabic (hereinafter SNA), on which there appears to be little previous study.

First, this chapter will review previous work in the literature that investigated the null-subject parameter with respect to dialects that exhibit diversity in the NSL's properties. Next, the background of the SNA dialect will be presented. Following this, a new data set featuring SNA will be introduced, examined, and discussed. The null-subject parameter of SNA will be compared and contrasted with that of SA to determine whether there are any syntactic

differences between the Saudi Najdi dialect and SA. Finally, conclusions will be drawn about SNA's adherence to NSL properties, and about the status of SNA as a NSL.

3.2 Literature Review

The null-subject parameter of Saudi Najdi Arabic has received little previous attention by researchers. Instead, this section will review previous scholarship, which compared the null-subject parameters of NSL dialects.

Camacho (2013) discussed how the properties of NSLs may vary in dialects of a single language, just as they vary from one language to another. He used Dominican Spanish and Brazilian Portuguese as examples. Spanish and European Portuguese, the respective parent languages of each dialect, are both NSLs. However, he found that both dialects had taken on features that did not correlate with NSL properties. These dialects can be considered to be shifting toward tendencies of NNSLs, though both still make use of null subjects.

According to Camacho, Dominican Spanish lacks free inversion in wh-questions, permits dummy pronouns (overt expletives), and does not violate the that-trace filter. Dominican Spanish examples are seen below in (38) .

(38) [7]

a. ¿Qué tú compraste?
what you bought
"What did you buy?"

b. *Que compraste tu?
what bought you
"What did you buy?"
(from Camacho, 2013)

c. Te dormiste.

[7] c and d were obtained via personal correspondence with native speakers.

you sleep
"You are asleep."

d. *Dormiste te.
sleep you
"You are asleep."

As seen in example (38) , Dominican Spanish does not allow free inversion in either wh-questions or simple sentences. As established in chapter 2, lacking free inversion is a trait of NNSLs. Ordóñez and Olarrea (2006) claimed the lack of inversion in wh-questions is constrained by person and number. The pronoun *tu* is a clitic pronoun and must be moved to TP; the subject will be in the same position or it could move to Comp. Example (39) exhibits an overt resumptive, or dummy, pronoun in Dominican Spanish.

(39)

En el campo ello sí hay mucho que hacer.
in the field there yes have much that do
"In the fields there is much to do."

According to Camacho (2013) the presence of the overt resumptive pronoun *ello* suggests two hypotheses: first, that Dominican Spanish is adapting and shifting to NNSL tendencies, and second that some of the properties ascribed to this non-null-subject parameter seem to cluster together.

In contrast to Dominican Spanish, Modesto (2000) states that Brazilian Portuguese now lacks null subjects in most cases. He found that the dropped pronouns must be locally bound by a c-commanding subject or by a bound object. See example (40) below.

(40)

a. O Pauloi convenceu o Pedro$_j$ que proi/$*_j$/$*$k tinha que ir embora.
the Paulo convinced the Pedro that pro had to go away
"Paulo convinced Pedro that he had to go away."

b. O Pauloi convenceu o Pedroj que elei/j/k tinha que ir embora.

the Paulo convinced the Pedro that he had to go away
"Paulo convinced Pedro that he had to go away."
(Brazilian Portuguese, Modesto, 2000)

Modesto argues that in example (40)a, the dropped pronoun is bound by the proceeding c-commanding subject, and not with a preceding *in situ* object. In example (40)b, the overt subject *he* can be considered referential with the object, or with a third referent.

Kenstowicz (1989) conducted a brief comparison study between two Arabic dialects, Levantine and Bani-Hassan, with regard to the null-subject parameter. Levantine Arabic (LA) is spoken on the eastern coast of the Mediterranean in Lebanon, Israel, the Palestinian territories, and parts of Syria and Jordan. Bani-Hassan (BHA) is a dialect spoken by a Bedouin tribe of the Jordanian desert. Kenstowicz's study of these two dialects was focused on examining the occurrence of null subjects in subordinate clauses.

According to his findings, LA differs from BHA primarily with regard to the that-trace effect, which has consequences in LA's treatment of the resumptive pronoun and free inversion. LA exhibits the same characteristics with respect to *that* clauses as English.[8] The complementizer "innu" in LA acts in the same manner as "that" in English; a subject NP may only be extracted from a *that* clause when the complementizer is omitted. This behavior is illustrated in example (41).

(41)

a. ʔayy fustaan$_i$ Fariid kaal innu l-bint ištarat e_i?
Which dress did Farid say that the girl bought?

[8] The author wishes to make it clear that the claim that LA patterns with English with regard to *that* clauses comes from Kenstowicz. How *innu* clauses may relate to this work's interpretation of SA *that* clauses is beyond the scope of this work.

b. ʔayy bint$_i$ Fariid kaal inn-ha$_i$ ištarat l-fustaan?
Which girl did Farid say that-she bought the dress?

c. ʔayy bint$_i$ Fariid kaal e_i ištarat l-fustaan?
Which girl did Farid say bought the dess?

d. *ʔayy bint$_i$ Fariid kaal innu e_i ištarat l-fustaan?
Which girl did Farid say that bought the dess?
(from Kenstowicz, 1989)

In (41)a and (41)b, a form of the subject is realized after the complementizer *innu*, and the resulting sentence is grammatical in LA. As shown in (41)b, the subject can appear in the form of a clitic attached to the complementizer, which plays the role of a resumptive pronoun.[9] The sentence of (41)c is grammatical because the complementizer is omitted. (41)d, however, is ungrammatical because the complementizer is immediately followed by the verb rather than the subject. Kenstowicz emphasizes that while LA permits missing subjects in main clauses, the omission of a subject pronoun following the complementizer *innu* will result in an ungrammatical sentence: the subject must be realized at least as a clitic, as in examples (41)b and (42)b.

(42)

a. (hiy) ištarat l-fustaan.
(she) bought the dress.

b. Fariid kaal inn-ha ištarat l-fustaan.
Fariid said that-she bought the dress.

[9] As a non-null subject language, English does not allow free use of resumptive pronouns in wh-questions as LA does. This is the single aspect of the that-trace effect with which English and LA do not correlate.

Due to the constraint that a subject must appear immediately following the complementizer *innu*, free inversion is impossible in *that* clauses in LA.[10] Kenstowicz provided the example in (43) to illustrate. In (43)b, inversion is attempted, breaking the constraint and resulting in an ungrammatical sentence.

(43)

a. Fariid kaal innu l-bint ištarat l-fustaan.
F. said that the girl bought the dress.

b. * Fariid kaal innu ištarat l-bint l-fustaan.
F. said that bought the girl the dress.

Kenstowicz found the BHA dialect to differ from LA with respect to the null-subject parameter. In BHA, subjects and objects can both be freely extracted from *that* clauses in wh-questions, as illustrated in example (44). The object is extracted in (44)a, while the subject is extracted in (44)b. Both sentences are grammatical in BHA, while the sentence of (44)b would be ungrammatical in LA.

(44)

a. wayy libaas$_i$ Fariid gaal innu al-binit ištarat e_i?
which dress did F. say that the girl bought?

b. wayy binit$_i$ Fariid gaal innu e_i ištarat al-libaas?
which girl did F. say that bought the dress?

It is acceptable in BHA, as in LA, to omit the complementizer *innu* and the subject, seen in (45)a, as well as to use a resumptive pronoun in the form of a clitic attached to the complementizer, as in (45)b.

[10] This is also true of ʔanna clauses in SA as well, due to the restraint discussed in Chapter 2 that ʔanna must be followed by an NP.

(45)

a. wayy binit$_i$ Farid gaal e_i ištarat al-libaas?
Which girl did F. say bought the dress?

b. wayy binit$_i$ Fariid gaal inn-ha$_i$ ištarat al-libaas?
Which girl did F. say that-she bought the dress?

Unlike LA, however, BHA allows missing subjects in *that* clauses. This is shown in example (46)a. Furthermore, free inversion in that clauses is also acceptable. Free inversion is illustrated in (46)b and (46)c.

(46)

a. Albinit gaalat innu ištarat al-libaas.
the girl said that bought the dress.

b. Faarid gaal innu al-binit ištarat al-libaas.
Farid said that the girl bought the dress.

c. Fariid gaal innu ištarat al-binit al-libaas.
Farid said that bought the girl the dress.

The literature discussed in this section all supports a general conclusion that the properties of NSLs can vary between dialects of single languages in the same manner as they do from language to language. Camacho discussed how two Spanish dialects, over time, are mimicking English and shifting toward NNSL tendencies. Kenstowicz's findings indicated that LA is not adapting some of the null-subject parameter, and instead mimics the English setting, while BHA exhibits all properties of the null-subject parameter without any constraints with respect to the occurrence of null subjects in subordinate clauses.

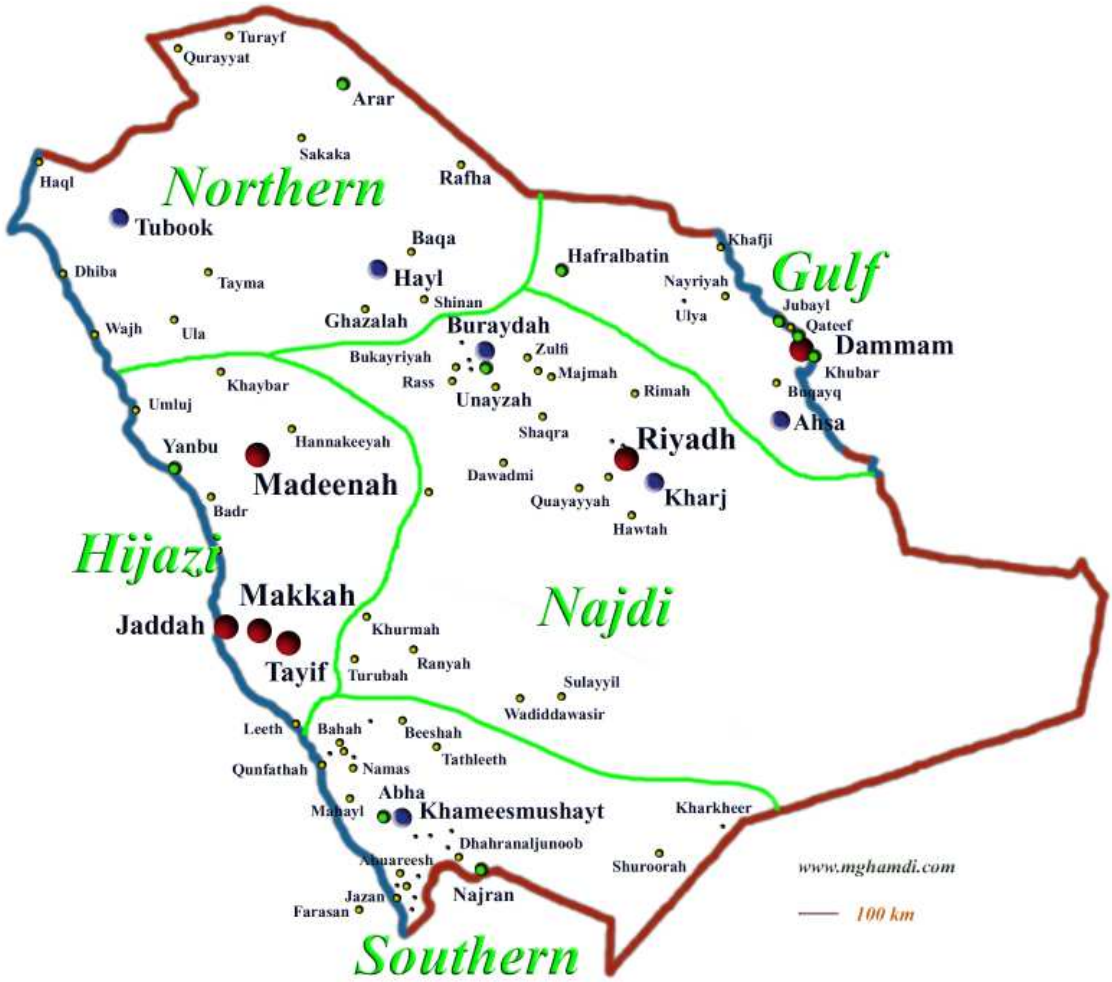
3.3 Saudi Najdi Arabic

Before discussing SNA in the context of the null-subject parameter, it is important to give a brief linguistic background of the characteristics of SNA. In this section, linguistic features that distinguish this dialect form other Arabic dialects will be presented.

SNA is one of the major SA varieties spoken in Saudi Arabia. Of 9,977,000 total registered speakers, approximately 8,000,000 reside in Saudi Arabia (Lewis et al., 2014). In Saudi Arabia, Najdi is commonly used in the spoken media, and is thus a common dialect throughout the country. It is regarded by its speakers as a prestigious dialect because it has preserved most of the SA features (Ingham, 1994).

The general regions of each major dialect in Saudi Arabia are shown in the map in (47). There are no scientific studies that have set an accurate geographic region for SNA; however, it is generally agreed that its speakers are scattered from the borders of Alkharkheer in the south to Hayl in the north, and from Ahsa in the east to the Hijaz Mountains in the west. SNA is known to be the dialect of the desert region in Saudi Arabia (Al-Sweel, 1981).

(47)



There are four subdialects of Najdi. Badawi, or nomad Najdi, is spoken by the tribes and villages of the Najdi region of Saudi Arabia. Central Najdi is spoken mainly in the capital city of Riyadh and the towns that surround it. The Qassimi dialect is spoken primarily in northern Riyadh city. Finally, Southern Najdi is spoken in the southern areas of the Najdi region (Ethnologue, 2014). This work will concentrate on the Central Najdi dialect.

3.3.1 Features of Najdi dialect

3.3.1.1 SNA phonological features. SNA has special features that distinguish it from the vast number of other Arabic dialects. In this section, some general characteristics of SNA will be

presented. To start, phonetics will be discussed. Lewis (2013) adapted Al-Sweel's (1981) phonemic inventory of SNA. According to them, there are 23 consonant sounds produced in this dialect, shown in Table 1 below, and ten vowels, shown in Table 2.

Table 1: SNA Consonant Sounds

		Bilab	Interdent	Alveolar	Palatal	Velar	Uvular	Pharay	Glottal
Stops	Voiceless			t T		k			ʔ
	Voiced	b		d D	z	g			
Affricates	Voiceless			ʦ					
	Voiced			ʤ					
Fricatives	Voiceless	f	th	s S	sh		x	ħ	h
	Voiced		TH	z			ɣ	ʕ	
Nasals		m		n					
Liquids				l r					
Glides					y	w			

(cf. Lewis, 2013 and Al-Sweel 1981)

Table 2: SNA Vowels

		Front (un-round)	**Central**	**Back (round)**
High	long	i:		u:
	short	i		u
Med		e:		o:
	short	e		o
Low	long		a:	
	short		a	

(cf. Lewis, 2013 and Al-Sweel, 1981)

One example of a consonant sound that is unique to the SNA dialect is the affricate voiceless alveolar *ts* sound. SNA has palatalized the voiceless velar stop [*k*] sound from SA into a */ts/* sound. In phonology this phenomenon is referred to as palatalization, which is a cross-linguistically common phonological process in which consonants are usually palatalized when they occur before high front vowels */i/*. For example, the SA word /kitha/ (which means *this*) is pronounced differently in SNA; it is pronounced as */tsetha/*.

3.3.1.2 Types of pronouns in SNA. Another feature that distinguishes this dialect from other Arabic dialects is its pronouns. SNA has three types of pronouns. The first two are strong and weak personal pronouns. What distinguishes them is that strong pronouns are phonologically stressed, and independent and weak pronouns are dependent, frequently appearing as clitics. These two types **w**ere categorized by Lewis (2013). According to him there are eleven strong pronouns and ten weak pronouns in SNA. Any pronoun from either set can be dropped or null depending on the syntactic structure of the sentence, which will be discussed in the following section. Table 3 illustrates SNA strong pronouns and Table 4 shows its weak pronouns.

Table 3: SNA Strong Pronouns

		SG	**PLU**
1	FEM/MASC	anna	Inna
2	MASC	ant	antum/ antu
	FEM	anti	antu/antin
3	MASC	hu	hum
	FEM	hi	hum/hin

(cf. Lewis, 2013)

Table 4: SNA Weak Pronouns

		SG	PLU
1	FEM/MASC	-ni	-na
2	MASC	-ik	-kum
	FEM	-ki	-kin
3	MASC	-ih	-hum
	FEM	-ha	hum/hin

The weak pronouns of SNA appear in different positions and contexts. Sometimes they come in the form of a clitic attached to verbs. They may also take the syntactic structure of an object after subject agreement. Finally, they may play the role of a possessor. In sentences such as (48)a, there are no weak pronouns because the object "Mansour" surfaces as a DP. However, in example (48)b the weak pronoun *–ih* is cliticized after the verb and adopts the role of an object after subject agreement. In (48)c, the weak pronoun plays the role of a possessor in DP (Lewis, 2013).

(48)

a. ʔarif-t mansour.
know-1psg mansour.
"I know Mansour."

b. ʔarif-t-ih.
know-1sg-him
"I know him."

c. raħ-t la-h
go-1psg.past to-him
"I went to him."

In addition to Lewis' (2013) two types of pronouns, null subjects or pro-drops (discussed in Chapters 1 and 2) can be considered the third type of pronoun. SNA preserves this type of

pronoun from SA. A data set will be presented as well as discussed in the following section of this thesis.

3.4 Data and Analysis

Evidence from the literature has now been presented, suggesting that the null-subject parameter of NSL dialects can exhibit variation, and the SNA dialect has been introduced. Now, light will be shed on the core of this chapter, which is focused on the null-subject parameter of SNA. The intent of this chapter is to determine whether the null-subject parameter of SNA differs in any way from that of SA with regard to the five properties introduced and discussed in Chapter 2.

Each subsection of this section will present SNA data for one of the five null-subject parameter properties discussed in the previous chapter, and state a conclusion about the correlation of SNA and SA. For the sake of brevity, the definitions and significance of the properties will not be restated here; refer back to section 2.2 for detailed discussions of each property.

3.4.1 Free inversion in SNA. As in SA (see section 2.2.4), LA, and BHA (see section 3.2) SNA exhibits free inversion. This is illustrated in (49).

(49)

a. Ahmed fataħ el-bab
Ahmed opened the door
"Ahmed opened the door."

b. Fataħ el-bab Ahmed.
opened3p.sg.m the door Ahmed
"Ahmed opened the door."

The sentences in (49) are the SNA equivalents of (17) in section 2.2.4. The SV and VS word orders necessary for free inversion are both grammatical. The availability of VOS order in

SA, which is considered to be the classic example of free inversion (seen in (49)b) is also available in SNA. The only differences between SA and SNA here are slight phonological changes, which have no influence on syntactic structure. It can thus be concluded that SNA exhibits free inversion as in SA.

3.4.2 Missing subjects in SNA. Missing subjects in SNA are illustrated in (50) below. These examples are SNA equivalents of the SA examples in (10).

(50)

a. qara-t kta:b
read-3p.s.fem a book
"She read a book."

b. kta:bet riwaya
wrote-3p.s.fem a novel
"She wrote a novel"

c. wasʿalu lbariħa
arrived-3p.pl last night
"They arrived last night."

As can be seen, SNA mimics SA completely in terms of missing subjects. It upholds the same standard of pro licensed by AGR features under INFL. The differences between the two example sets are merely phonological.

3.4.3 *That-trace filter in SNA. In section 2.2.5, it was shown that SA is a very interesting case of an NSL, which seems to obey the that-trace filter and parallel English, but in actuality this seeming violation comes from an independent property of the language stating that the complementizer *ʔanna* must be followed by some accusative NP, which may be a subject or an object. This principle is defined in (25). Recall the conclusion that SA does, in fact, exhibit a violation of the that-trace filter, and should be seen as patterning with other NSLs in this regard.

If SNA is again to parallel SA, the expectation is that following *anna* with a trace will be ungrammatical in SNA. This is seen in (51), examples similar to the SA examples of (23).

(51)

*man taʔtaqid [CP ***anna*** [IP ***t*** [fataħa l-bab-a]]]
who believe-2ndSgMasc that opened the-door-Acc
*"Who do you think that opened the door?"

The expectation proves correct. Based on this example, SNA parallels SA in respect to the principle suggested in Chapter 2, (27). However, *anna* retains the same behavior as in SA. The principle of (25) must also be followed in SNA; that is, some overt NP must follow *anna*. This NP may be an object, which violates the that-trace filter because the filter expects a subject to follow the complementizer.

As an interesting aside, the mood assigner *ʔan*, discussed in section 2.3, does not exist in SNA. In conclusion, with regard to the that-trace filter, SNA continues to syntactically parallel SA. However, SNA shows a small syntactic variation from SA in its disallowance of the mood assigner *ʔan*.

3.4.4 Long wh-movement of subjects in SNA. Recall from section 2.2.8 that long wh-movement of subjects was the single property for which SA definitively did not parallel other NSLs. It instead patterned with English, an NNSL. As with the previous properties, SNA again mimics SA, as seen in (52) below.

(52)

a. ʔai kas dann-at wafa ekhtār-at?
Which cup think-3p.sg.past.M. wafa choose-3p.sg.F. past
"Which bag did she think Wafa chose?"

b. *ʔai kas dann-at al-ishaʔa wafa ekhtār-at?
Which cup think-3p.sg.past.F the-rumor wafa choose-3p.sg.F.past
*"Which cup did she think the claim wafa chose?"

c. *man taʔtaqid man ʃaf?
who wonder-2p.sg.m who saw
*"Who do you wonder who saw?"

A comparison of (52) and the SA example in (31) reveals that SNA parallels SA in its disallowance of long wh-movement of subjects. As in SA, there is no clear explanation at this point for why this lone property is not exhibited in SNA. It is a fascinating question worthy of future research.

3.4.5 Null resumptive pronouns in SNA. Recall from section 2.2.10 that this work contends that SA does exhibit and allow null resumptive pronouns. Example (53), an SNA reworking of SA example (36)a, illustrates the use of null resumptive pronouns in SNA.

(53)

[CP [IP haðihi hije eʃ-ʃeʒera-h lli [CP lama [IP janzil *e* matar [IP *e* tek̲h̲aẓẓar]]]]]
this is the-tree that when fall rain-3p.sg becomes green.
*"This is the tree that when it rains, becomes green." or more correctly
"This is the tree that when it rains, it becomes green."

In this final property, SNA again mimics SA syntactically. The differences between the dialect and SA are again only phonological.

3.5 Conclusion

This chapter has shown that dialects of a single language may differ significantly from one another with regard to the null-subject parameter. Two previously studied Arabic dialects, Levantine and Bani-Hassan, were reviewed in section 3.2. These dialects showed slight differences in the null-subject parameter properties, both from each other and Standard Arabic. Saudi Najdi Arabic, however, has been shown to almost completely parallel Standard Arabic with regard to syntactic properties. It differs from SA only in its lack of the mood assigner *ʔan*, and in phonology.

Chapter 4: Computational Analysis of Arabic Pro-Drop — Implementation

4.1 Introduction

The previous two chapters have established the syntactic properties associated with Arabic pro-drop. Now, focus shifts to investigating the circumstances where pro-drop occurs in Arabic. Of particular interest is the optionality of pro-drop. Chapter 2 established that rich agreement in Arabic is what licenses the availability of pro-drop; this rich agreement makes it syntactically possible to drop subjects in almost any context. What, then, influences a speaker to omit or include a particular subject? This work adopts the standard assumption that the optionality of pro-drop is controlled more by discourse than by syntax.

One possible route to testing hypotheses about the optionality of pro-drop is by programmatically analyzing a large amount of Arabic text. Fortunately, this is made possible by corpora such as the OntoNotes Arabic treebank, a Standard Arabic corpus of nearly two million words (Maamouri et al., 2008). Pro-dropped subjects in this corpus were extracted and analyzed in various contexts.

This chapter is devoted to the technical details of the Python code used in the analyses. The next chapter will present and analyze the results of the experiments introduced in this chapter. This chapter is broken into three major sections. First is an explanation of why the Python language was chosen for this project. Next, the dataset is examined and terminology used throughout the rest of the chapter is introduced. Third, what is termed the "backbone" of the code

is discussed. Finally, the purpose and technical details of each experiment are presented. A short section on technical results (execution times, etc.) closes the chapter.

4.2 Justification

4.2.1 Why Python? The Python programming language was chosen to do the computational work of the project primarily because of its ease of use, high readability, Unicode support, and proven record of language processing and data visualization via third-party libraries. The potential downsides of using Python—primarily speed limitations—were determined not to be a deciding factor.

4.2.1.1 Ease of use. As a high-level programming language, an advantage Python holds over lower-level counterparts is ease of writing, which results in shorter development time. Python's level of abstraction means that the programmer does not need to be concerned with issues such as memory management.

4.2.1.2 Readability. A core tenet of Python's design philosophy is an emphasis on readability (Peters, 2004), on which a high priority was placed due to the collaborative nature of the project. Discussion about what a piece of code does, and how it does it, is made easier by a language like Python. The possibility that someone with a limited knowledge of programming will be able to follow such a discussion with ease is enhanced if the programming language can be read as pseudo-English. In addition, readability plays a part in shortening coding time, as quickly understandable code is easier to work with and expand upon.

4.2.1.3 Unicode support. A major factor in deciding how to proceed with the project was Unicode support. Unicode is a character encoding standard in the computing industry that provides support for many of the world's writing systems, including Arabic. In computing, a character is the smallest possible component of text. For example, a character is any individual

letter, digit, or symbol on a keyboard that has a visible representation on the screen when typed. An encoding standard dictates how a computer knows which character is which by assigning each character a specific numeric value.

Unicode contrasts with the American Standard Code for Information Exchange, or ASCII, which was the standard character encoding in the United States for many years, the most common on the World Wide Web until 2007 (Dubost, 2008), and is still the default in many applications. The foremost limitation of ASCII is that each character is allotted 7 bytes, meaning there are only 128 possible characters. This space is filled with the lower and uppercase English alphabet letters, digits, symbols, and text structure characters such as line breaks. Characters for languages beyond English are not representable with ASCII. In contrast, Unicode has the capacity for over a million separate characters (Unicode Consortium, 2014).

A programming language that facilitates easy and accurate reading, understanding, and writing of Arabic characters via the Unicode standard was a requirement for the project. Although Python has supported the use of Unicode to some degree since version 2.0, and it was greatly improved with the latest major release of Python, version 3. With the Python 3.x series, the default encoding of Python code files became UTF-8, a popular variant of Unicode, allowing easy use of any Unicode character in a code file. In addition, the default encoding of strings, a data type representing a sequence of characters, is also Unicode, so no special code is needed to work with, in this case, Arabic characters or words (van Rossum, 2009).

4.2.1.4 Third-party libraries. The final factor that went into the choice of Python is its proven record of high-quality third-party libraries, as well as its frequent adoption and use in the field of natural language processing. Two libraries in particular were noted prior to starting the project. First, the Natural Language Toolkit (NTLK) provides tools and interfaces for working

with language data. Second, matplotlib provides easy to use yet robust tools for data visualization in Python.

4.2.1.5 Potential limitations. Speed is the fundamental disadvantage that comes with use of any high-level interpreted language, as compared to a lower-level compiled language. The increased abstraction of a language like Python results in more time spent converting commands to a language a computer can understand and execute. When extremely large sets of data are being dealt with that could exhaust a computer's physical memory, memory management is another potential issue.

Both of these issues were determined to not be of consequence in this project. The size of the entirety of the data files examined is 30MB, divided into individual files smaller than 100KB. This is far below any reasonable threshold of concern for physical memory on any home computer made in the past fifteen years. Computation speed was also not of concern. A tree, such as a syntactic parse tree, is a very efficient structure often used in computing. Building, traversing, and searching the approximately twelve thousand trees used in this project is relatively cheap computationally. Indeed, full analysis of the entire dataset ended up taking approximately nineteen seconds on a mid-range desktop system.

4.2.2 Why a new tool? The data corpus used, OntoNotes, provides a relational database tool with a Python API. The use of this tool was considered, but the decision was made to create a new tool for multiple reasons. First, upon looking through the documentation of the OntoNotes tool it was felt that there was logical space for a new "intermediary" tool, containing a subset of the features of the OntoNotes tool, but with a shallower learning curve. Second, the OntoNotes tool is compatible only with the increasingly obsolete Python 2. A tool built specifically for

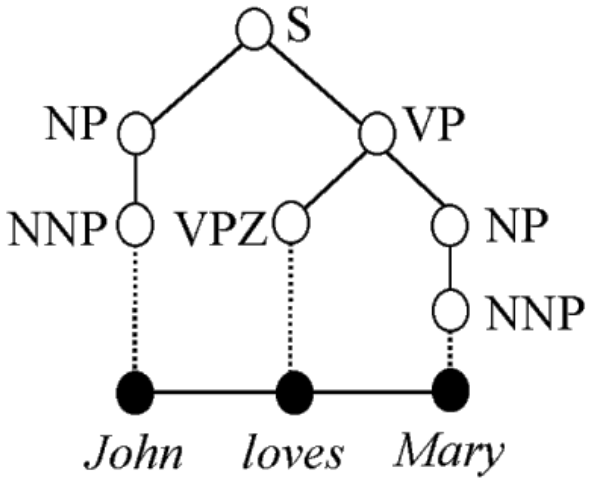
Python 3 could take advantage of its exclusive features, and possibly aid upcoming researchers already familiar with this version of the language.

4.3 Dataset

4.3.1 Terminology. Throughout the remainder of this chapter, particular terminology regarding the data in .parse files, and the trees they represent, is assumed; such terminology will be defined in this section. Terminology will be defined with regard to a simple English tree as an example. This tree is shown both in Treebank notation and as a traditional tree in (54) below.

(54)

```
1 (TOP (S (NP (NNP John))
2         (VP (VPZ loves)
3             (NP (NNP Mary)))
4         (PUNC .)))
```



Right image credit: Wikipedia

A *node* is defined as any piece of the tree structure in the image on the right marked with an open circle. A node can be thought of as an object with a specific location in a tree structure that is a container for more information. Every node has a *tag*, which is the string of capitalized letters immediately following the opening parenthesis in the notated form, indicating the part of speech (and associated semantic/syntactic information). The tags in this case are TOP (not shown in the tree image), S, NP, NNP, VP, VPZ, NP, NNP, and PUNC (also not in the right image). Tags can be lengthy and complex, and include symbols such as the addition sign, colons, and underscores. However, a tag is the entirety of any text unbroken by a space following an opening

parenthesis. *Base tag* is the term used to indicate a tag stripped of all modifiers and extraneous data, essentially at its most basic part-of-speech form. In the case of the IV3FS+IV+IVSUFF_MOOD:I tag in (55), the base tag is IV.

Some nodes have a *word* attribute. These nodes always occur at the end of a tree branch. The words in the example tree are "John," "loves," "Mary," and the punctuation period. In notated form, a word always occurs following a single space placed after a tag. Because words are always attached to end nodes, a word is always followed in notated form by a closing parenthesis. It is important to note that the filled-in circles in the tree image, each associated with a word, do *not* represent separate nodes. A dashed line indicates the words belonging to the nodes they are linked to.

Every node, with the exception of TOP, has a single *parent*. A parent is the attached node found immediately above a particular node. In the example tree, the parent of the VP node is the S node, while VP is the parent of both a VPZ and an NP node. A node may have many *ancestors*, which are any nodes visited when tracing back through the tree, starting at a particular node. For example, the ancestors of the NNP node with the word "Mary" are NP, VP, S, and TOP. A node may also have multiple *children*. A child is an attached node directly below a node in the tree. In the example, NP and VP are the children of S, but VPZ is not a child of S, despite occurring below S in the tree, as it is not directly attached. A node such as VPZ in the example is referred to as a *nested child* of S.

Nodes with no words are referred to as *through nodes*, because they are passed through when following the branch of a tree. Nodes with a word are *end nodes* or *leaf nodes*. A node with multiple children, such as VP or S in the example above, may be referred to as a *block node*, as the end result of a trace of the node's children results in a block of text.

4.3.2 The .parse format. The project was concerned with the portion of the OntoNotes Arabic Treebank data that have files with a .parse extension. In total, 599 of these files were examined. Each file contains multiple parse trees notated in a consistent manner. A simple Arabic tree in a .parse file is shown in (55) below.

(55)

```
1   (TOP (SQ (CONJ وَ-)
2            (VP (IV3FS+IV+IVSUFF_MOOD:I -تَنْظُرْ)
3                (NP-SBJ (-NONE- *))
4                (PP-CLR (PREP عَلَى-)
5                        (NP (PRON_3FS -هَا))))
6            (PUNC ؟)))
```

An important note about the .parse format is the use of parentheses to denote the beginning and end of single nodes or a block of nodes. The line breaks and indentation of these files is, likely, strictly for the benefit of being more human-readable. When reconstructing these trees programmatically, the structure can be determined entirely from the opening and closing parentheses, and the line breaks and whitespace can be ignored.

Python tests run on the entire dataset verified that every new tree begins with a TOP tag on the start of a line following a blank line (or at the head of the file), and that this was the single context in which a TOP tag appears. Programmatically, this made separating the notated trees a simple matter of breaking the files into chunks, starting with `(TOP` and ending at a blank line.

4.3.3 Pro-drop notation. Pro-drops are easily identifiable in the .parse file notation. The basic form of a pro-dropped subject is `(NP-SBJ (-NONE- *))`. The simple tree shown in (55) includes a pro-dropped subject. Slight variations of this format are possible: indices can be appended to the NP-SBJ tag with either an equal sign or a dash, and to the asterisk indicating a pro-drop with a dash. These considerations mean that `(NP-SBJ=6 (-NONE- *-3))` is a valid representation of a pro-drop. The -NONE- tag on the pro-drop node itself is constant, although it

is not exclusive to pro-drops. It also appears on other empty category nodes such as traces. The combination of the -NONE- tag followed by the single asterisk is key to indicating a pro-drop. A trace, for example, would be represented by `(-NONE- *T*)`.

A pro-drop node, e.g. `(-NONE- *)`, may appear nested inside tags other than NP-SBJ, but these are not within the scope of this project. As the focus of this project was pro-dropped subjects, pro-drops inside NP-OBJ or NP-DTV nodes were not considered.

4.3.4 Verb notation. A verb is more difficult to identify than a pro-drop in .parse file notation. There is no single tag or block tag that represents a verb, and the placement of a verb within a verb phrase (VP) is not necessarily consistent.

Tags for verb nodes vary wildly. A variety of complex modifiers can be attached to the tag of a verb's node. Examples of verb tags found in the corpus are IV3FD+IV+IVSUFF_SUBJ:D_MOOD:I, PV+PVSUFF_SUBJ:3FD, and IV3MP+IV+IVSUFF_SUBJ:MP_MOOD:I. The information these modifiers represent is not relevant to the task of extracting verbs from the data, but make the exhaustive extraction of verbs more challenging. Fortunately, the opening characters of a verb node's tag consistently represent the base tag. As can be seen in the example tags above, testing if a tag begins with verb base tags such as "IV" or "PV" is enough to positively identify a primary verb node.

Moreover, the placement of verbs is not entirely consistent. There are constants, however. They always appear within a verb phrase (given by VP in treebank notation) and in the Arabic treebank must always precede the subject. In treebank notation context, "precede" means the verb will be written somewhere above the subject. Subjects that precede a verb in the original sentence have an NP-TPC tag when notated, and a trace node inside an NP-SBJ node is placed following the verb (Bies & Maamouri, 2003). The most common placement case is seen in the

simple tree example above. A verb phrase opens with `(VP` and immediately following is the verb, `(IV3FS+IV+IVSUFF_MOOD:I -تُسَيْطِرُ)` and then the subject. However, any number of nodes could exist as siblings of the verb and NP-SBJ either preceding or following the IV node.

4.4 Backbone

Prior to doing the actual pro-drop analysis, the fundamental "backbone" of the project had to first be coded. A well-written, well-tested, and robust backend was necessary to provide a proper environment for analysis scripts that are easy to read, and which contain as little repetitive "boilerplate" code as possible. A primary goal was to abstract large tasks—such as iterating through every tree in every .parse file and building Python objects from them—into simple functions callable by the analysis code. Another important goal was to provide safe and well-tested modules to the analysis code, so that concern over the accuracy of data being gathered and reported was not necessary.

4.4.1 The *parsetree* module. The first coding step was to implement Python objects representing both the nodes of a linguistic parse tree, as well as an object representing an entire tree that facilitates robust searching.

4.4.1.1 Parse tree nodes. The first piece of code to be discussed is that which creates objects representing nodes of parse trees, as discussed in section 4.3. Nodes can be divided into two fundamental types, through nodes and end nodes. A through node has at least one child, while an end node has a word attribute and no children. These two node types share common attributes, however: they both always contain a parent and a tag. These relationships were defined by creating an abstract base class called `ParseTreeNode` to define the shared attributes, and then the `ParseTreeThruNode` and `ParseTreeEndNode` classes to define the behavior and attributes that differ between the two types. Most attributes (parent and children

in particular) were defined as read-only so that the structure of the tree could not be altered after construction, whether accidentally or deliberately. Simple Boolean (effectively "yes or no") utility attributes `has_children` and `is_end` were made available on each node object to quickly and readably determine the node style without the need for explicit type-checking.

The `ParseTreeNode` class requires a "parent" argument when it is created. A new instance of the class, i.e. any new node object, automatically adds itself to its parent's children attribute upon creation. This creates the links in the tree structure automatically upon creation of each node, preventing possible errors elsewhere. An exception is the initial TOP node that begins each tree. It has no parent, so allowance is made for its parent attribute to be the special Python `NoneType` object, given by `None` in-code. The definition of the `ParseTreeEndNode` class is shown in (56) below.

(56)

```
102  class ParseTreeEndNode(ParseTreeNode):
103      """
104      ATTRIBUTES:
105        * has_children (read-only)
106        * is_end (read-only)
107        * word
108
109      INHERITED ATTRIBUTES:
110        * parent (read-only)
111        * tag
112      """
113      def __init__(self, parent, tag, word):
114          super().__init__(parent, tag)
115          self.word = word
116
117      @property
118      def has_children(self):
119          return False
120
121      @property
122      def is_end(self):
123          return True
```

4.4.1.2 Parse tree objects. With the code for the individual nodes complete, the next step was an object representing an entire tree. These objects are defined by the `ParseTree` class.

The purpose of the class is to provide methods for traversing and searching a tree's nodes, as well as facilitate conversion of treebank notation into a usable Python `ParseTree` object.

Rather than redundantly store every node contained in the tree it represents, `ParseTree` objects contain only a reference to the topmost node in the tree. The tree structure can then be traversed using the children and parent attributes described in section 4.4.1.1 above.

Several traversal methods were implemented for ease-of-use and efficiency purposes. The basic traversal method is `iternodes`. The method yields every node in the tree in depth-first order. In the case of syntactic parse trees, depth-first order means that if each word was printed as it was visited, the sentence represented by the tree would be printed in its proper order. This method is a generator, meaning it yields one item at a time and is used in a manner such as `for node in iternodes(): [do something]`. The method was implemented recursively, repeatedly calling itself with updated arguments, seen at line 248 of (57), until an endpoint was reached. In this case, recursion stops when the rightmost end node has been reached.

(57)

```
238     def iternodes(self, **kwargs):
239         """
240         Yield each node during depth-first traversal of tree.
241         """
242         # node defaults to self.top when method called with no arguments
243         node = kwargs.get('node', self.top)
244         yield node
245 
246         if node.has_children:
247             for child in node.children:
248                 for n in self.iternodes(node=child):
249                     yield n
```

More specific traversal methods include `iterendnodes` and `iterwords`, which yield only end nodes and the words of end nodes, respectively. These methods offer an efficient way to deal only with end nodes when needed (such as when searching by word, which only end

nodes can contain). Without these methods, time could be wasted repeatedly traversing an entire tree when only specific nodes are desired. This was accomplished by caching a list of references to end nodes when the tree was created. This caching behavior could be disabled, which is only recommended in specific circumstances described in the documentation of the `parsetree` module.

With methods for traversing the parse trees complete, the next step was to implement searching.

4.4.1.3 Parse tree searching. A prerequisite to the intended analyses was the ability to search a parse tree for nodes that meet specific criteria, such as pro-dropped subject nodes. The aim was to create a robust mechanism to allow various search types on each node attribute. A well-designed search system would enable fine-tuned control of what nodes were being studied in the analysis stage, without the need to make large changes to existing code or write new code.

Parse tree searching is accomplished exclusively via the `ParseTree.search` method. The node attributes on which specific searches can be performed are tag, parent tag, and word. For each type of searchable attribute, the method accepts two arguments: the search query string and a named flag denoting the type of search to perform with that string. A search can be performed on any combination of attributes.

The search type flags are `CONTAINS`, `CUSTOM`, `EXACT`, `IS_NOT`, `REMATCH`, `NOT_REMATCH`, and `STARTSWITH`. Each denotes how a search on an attribute will be performed. `CONTAINS`, `EXACT`, `IS_NOT`, and `STARTSWITH` simply match the passed query string against a node argument's value in the manner described by their names. `REMATCH` and `NOT_REMATCH` expect a regular expression as the query string. Regular expressions, used often in computing, are highly capable at complex pattern-matching. Any desired search pattern would

likely be representable in regular expression form, so these flags provide a powerful fallback when none of the simpler flags are sufficient. The `REMATCH` flag considers any node attribute for which `re.match` (`re` is Python's built-in regular expression library) returns a value other than `None` as a match. `NOT_REMATCH` works the opposite way, and is useful for using the same regular expression pattern to get complementary results. The `CUSTOM` flag further allows fine-tuned control by accepting a user-provided function to perform the search validation.

Each flag, with the exception of `CUSTOM`, could be concisely written as a lambda expression. A lambda expression acts as an anonymous function able to be defined inline. Each flag's lambda function has the same signature, accepting a search phrase followed by the value of an actual node attribute, and is expected to return `True` or `False`. A private method, `_get_comparison_function`, is called for each attribute and returns a Python function object used to perform the necessary matching. This behavior enabled the search methods to use each returned comparison function in a single manner, so all searching is accomplished in a single block of code, rather than with branching behavior for each flag/search type. The lambda expressions for each flag are shown in (58) below.

(58)

```
347         funcmap = {
348             self.EXACT       : lambda phrase, s: phrase == s if phrase else True,
349             self.CONTAINS    : lambda phrase, s: phrase in s,
350             self.STARTSWITH  : lambda phrase, s: s.startswith(phrase),
351             self.REMATCH     : lambda pattern, s: re.match(pattern, s),
352             self.NOT_REMATCH : lambda pattern, s : re.match(pattern, s) is None,
353             self.IS_NOT      : lambda phrase, s: not phrase == s,
354         }
```

An important note about the behavior of the `search` method and the lambdas is that a comparison function is run on every attribute during every search, whether it is specified by a user to be included in the search or not. However, the values for non-provided search strings

default to empty, and the default comparison is `EXACT`. As shown in (58), the EXACT lambda expression always validates an empty string as a match. This behavior was designed to enable better abstracted and readable code, as it eliminated the need to explicitly check if search queries were empty. The end result was two simple and elegant private search methods that the master public search method could call, depending on whether all nodes or only end nodes were being searched. These methods return accurate results for any combination of queries and flags sent to the search method, while being abstracted enough to not be concerned with the specific contents of the arguments. The `search` method and the end node method, stripped of comments and documentation for the sake of brevity, are shown in (59).

(59)

```
261     def search(self, tag='', word='', tag_flag=0, word_flag=0, **kwargs):
262         tagfunc = self._get_comparison_function(tag_flag, 'tag', **kwargs)
263         wordfunc = self._get_comparison_function(word_flag, 'word', **kwargs)
264 
265         parent_tag = kwargs.get('parent_tag', '')
266         parent_flag = kwargs.get('parent_flag', 0)
267         parentfunc = self._get_comparison_function(parent_flag,
268             'parent', **kwargs
269         )
270 
271         if word or word_flag in (self.CUSTOM, self.IS_NOT):
272             return self._search_end_nodes(tag, tagfunc, word, wordfunc,
273                 parent_tag, parentfunc
274             )
275         else:
276             return self._search_all_nodes(tag, tagfunc, parent_tag, parentfunc)
[...]
384     def _search_end_nodes(self, tag, tagfunc, word, wordfunc, parent_tag, parentfunc):
385         results = []
386 
387         for node in self.iterendnodes():
388             if (tagfunc(tag, node.tag)
389                 and wordfunc(word, node.word)
390                 and parentfunc(parent_tag, node.parent.tag)
391             ):
392                 results.append(node)
393 
394         return results
```

Example searches are described in this paragraph. If `search` is called with the `tag` argument “N” and the `tag_flag` argument `STARTSWITH`, any node that has a tag beginning

with the letter "N" is returned. Using the tree in (54) as an example, both NP nodes and both NNP nodes would be returned. If the same search was performed, but with the addition of the `word` argument "John" and the `word_flag` argument `IS_NOT`, only the NNP node with the word attribute "Mary" would be returned. The `IS_NOT` flag results in any nodes that do not have a `word` attribute exactly matching "John" are considered matches. The VPZ and PUNC nodes, which both have words that do not match "John," are not returned because of the restriction that the `tag` attribute must start with "N." Finally, the NP nodes, which were returned when the only search attribute was tag, are not returned when the `word` restriction is added. For efficiency purposes, if a word argument is passed, the search method only searches the nodes yielded by the previously described `iterendnodes`, limiting the search to only those nodes that have a `word` attribute. It was determined to be a reasonable assumption that a user filtering nodes based on the `word` attribute is not interested in nodes that have no such attribute.

The result of the search implementation is a system that is easily extensible and highly robust. The flag system for search types facilitates easy extensibility of searching. Adding a new search type involves only assigning a (unique) integer value to the named flag attribute, and mapping a function to that name, as seen in (58). Accepting a separate flag per searchable attribute allows for highly specific and refined search queries. The `REMATCH` and `CUSTOM` flags provide a user with powerful levels of customization, while the simpler flags aid in ease of use for more trivial searches.

4.4.2 Importing the data. The second major task of the "backbone" code was to read in the trees from the OntoNotes Arabic treebank .parse files and convert them to the Python `ParseTree` objects described in the previous section.

As mentioned previously, a .parse file can contain many parse trees. Extracting individual trees from the files proved a simple task, due to the consistency of the .parse format and the entire dataset. An individual tree in treebank notation begins with "(TOP" and ends at a blank line.[11] Sets of lines of the preceding criteria were extracted and sent as an argument to the constructor of the `ParseTree` class, which constructs its nodes from these lines. The *util* module was created to contain functions related to reading the .parse files. A function, `itertrees`, was written to automatically create `ParseTree` objects from each tree in a .parse file. An additional function, `itertrees_dir`, was written to do the same task for each .parse file in an entire directory.

The construction of the `ParseTree` object from treebank notation occurs in the `ParseTree._build_from_lines` method. This method scans the notated tree line by line, identifies nodes and instantiates them. As seen previously, the tree structure is created automatically with the instantiation of a node, assuming it receives the proper parent node. To identify a node, the method relied on two regular expressions, each representing either a through node or an end node. Put simply, a through node pattern is an opening parenthesis, immediately followed by a tag, then a space, then another opening parenthesis representing a child node. An end node is an opening parenthesis, a tag, a space, a word, and a closing parenthesis. Closing parentheses following an end node represent the end of an ancestor node, so the tree sets the pointer to the current node to that node's parent for each closing parenthesis found. In this manner the trees were accurately reconstructed. An exception (error) would be raised in the

[11] This was verified via testing to be true for every tree in the dataset.

event that the number of opening and closing parentheses did not match for a given tree, but no such error occurred when importing the OntoNotes Arabic treebank.

The end result is that a single function call will iterate through `ParseTree` objects created from each tree of each .parse file in a directory given by a path string. An operation could be performed on every tree in the entire treebank with the code `for tree in itertrees_dir('path/to/data/'): [do something…]`.

4.5 Analysis Code

The task of gathering metadata about the corpus data was accomplished by writing Python code that made use of the "backbone" code described in detail in section 4.4. The code described in this section was all placed in the *subjectverbanalysis* module. There were three major tasks to accomplish in order to gather the desired metadata. First, for every tree in the corpus, pro-dropped subjects must be identified. Second, non-pro-dropped subjects must be identified. Finally, a reliable and accurate procedure to determine the verb most closely associated with each subject, regardless of pro-drop status, was necessary.

4.5.1 Pro-dropped subject identification. The aforementioned parse tree node searching mechanism made identifying and extracting pro-dropped subjects a simple matter of providing the correct arguments to the `ParseTree.search` method.

As described in the section detailing pro-drop notation in OntoNotes treebank format, a pro-drop is represented in node form with a tag of "-NONE-" and a word primarily composed of a single asterisk. A single dash and an integer can immediately follow the asterisk.

The tag argument to `search` used the default exact match, as a pro-drop node tag is always "-NONE-" without exception. The parent tag argument for a pro-drop subject begins with "NP-SBJ" in every case, with possible trailing modifiers that are irrelevant for this purpose.

Thus, the `STARTSWITH` flag was applied to identify a valid parent. To account for the varying format of the word attribute of a pro-drop node, a regular expression allowing for the two possible patterns was provided as the search query string, and the `REMATCH` flag used. The final function used to return each pro-dropped subject is shown in (60).

(60)

```
PRODROP_WORD_PATTERN = '^\*(?:-\d+)?$'

################################################################################
def iterprodrops(tree):
    """
    Yield pro-drop nodes, i.e. (-NONE- *) nodes whose parent is a variant
    of NP-SBJ.
    """
    return (node for node in tree.search(
        tag='-NONE-',
        word=PRODROP_WORD_PATTERN,
        word_flag=tree.REMATCH,
        parent_tag='NP-SBJ',
        parent_flag=tree.STARTSWITH)
    )
```

4.5.2 Non-pro-dropped subject identification. Identifying non-pro-dropped subjects was a similar task to identifying pro-drop subjects, as many characteristics are shared between the two types of nodes. No restriction was placed on tag. The tag can be "-NONE-" because other empty categories beyond pro-drops utilize that tag. The parent tag must begin with "NP-SBJ" as in the pro-drop search. The word attribute is searched on the exact opposite criteria as for pro-dropped subjects. The same regular expression is given, but the `NOT_REMATCH` flag is sent to `search`.

4.5.3 Associated verb identification. The process of identifying the verb most closely associated with a subject was more difficult, but fortunately identical for both sets of subjects. As previously discussed in the Verb Notation section, the manner in which verbs appear in treebank notation is relatively inconsistent compared to subjects.

4.5.3.1 Verb tag considerations. The first decisions to be made regarding verb association revolved around exactly what tags constitute the sort of verbs desired. As discussed in the Verb Notation portion of the Dataset section, there is no single tag that represents "the" verb of a verb phrase, and tags that represent verbs often contain complex modifiers. Furthermore, many nodes irrelevant to the purpose of this study can exist above a subject node in a verb phrase.

Unfortunately, no complete reference of base tags found in the OntoNotes Arabic treebank could be found, which made an exhaustive list of valid verb tags difficult to create. To solve this problem, code was written to accumulate and report data about the siblings of NP-SBJ nodes, i.e., potential verb nodes. A Python dictionary mapped each unique sibling tag to a count of the number of its occurrences in the corpora. A report was printed with this data, which served as a suitable reference of every potential verb tag to be found in the corpora.

The NP-SBJ sibling report revealed two important pieces of data. First, common verb tags became immediately apparent due to their high rate of occurrence, as approximately 85% of NP-SBJ nodes have a verb node as a sibling. Second, it revealed the trend that verb tags seem to consistently begin with a base tag. It was found that despite tags containing a great deal of extraneous data, they invariably began with an identifier such as PV or IV. This was a relief, as it made identifying these nodes easier. It was later verified, via testing for the whole dataset, that any tag containing a verb base tag in any position also begins with a verb identifier.

The result of studying the NP-SBJ sibling report was a list of base tags for verbs. To be considered as a verb node associated with a subject, a node's tag must begin with IV (imperfect verb), PV (perfect verb), VERB, or PSEUDO_VERB.

4.5.3.2 Verb placement considerations. The general inconsistency of verb placement relative to a verb phrase and a subject is discussed in the Verb Notation portion of the Dataset section above.

The verb-placement standards limit the search area for an associated verb to nodes defined above a subject node in treebank notation. In the context of a depth-first traversal of a tree structure, verb nodes can be found in sibling nodes to an NP-SBJ node that are visited prior to the NP-SBJ node itself. The first implementation that identified associated verbs operated under this assumption, and found general success. Trees for which identifying an associated verb failed were reported by the program, and examination of these trees showed that an associated verb can sometimes appear as a sibling to an ancestor of a subject node.

The final implementation of identifying associated verbs, seen in (61) below, searched the siblings of a subject node, then the siblings of each of the node's ancestors, until either a verb was discovered or the start of a verb phrase was reached. The verb phrase limitation was used to reduce the possibility of false-positive associations, i.e., improperly reporting that a verb was associated with a particular subject. The improved implementation raised the rate of successfully discovering an associated verb for both pro-dropped and non-pro-dropped subjects by approximately 5%.

(61)

```
343     def _get_associated_verb(self, node):
344         """
345         Return nearest verb node to passed 'node' by searching the previous
346         siblings of 'node' and its ancestors until either a verb is found
347         or a VP node is reached.
348 
349         If no associated verb found, return list of tags of nodes visited
350         during search.
351         """
352         visited_tags = []
353 
354         while node.parent is not None and not node.tag.startswith('VP'):
355             result = self._check_siblings_for_verb(node)
356 
357             # Verb found in siblings. Return verb node
358             if hasattr(result, 'tag'):
359                 return result
360 
361             visited_tags += result
362             node = node.parent
363 
364         return visited_tags
```

4.5.4 Python implementation. The *subjectverbanalysis* module contains several classes useful to an end-user with identical APIs. These classes are `ProdropAnalyzer`, `NonProdropAnalyzer`, and `CombinedAnalyzer`. All three classes inherit from the abstract `BaseAnalyzer` class. `CombinedAnalyzer` inherits directly from it, while `ProdropAnalyzer` and `NonProdropAnalyzer` have an additional abstract parent `SubjectVerbAnalyzer` below `BaseAnalyzer` in the hierarchy. The exact nature of the abstracted functionality is beyond the scope of this document.

Each of the three classes define the methods `do_analysis`, `write_report_basic`, and `write_report_full`. ProdropAnalyzer and NonProdropAnalyzer contain counters and dictionaries that accumulate data when `do_analysis` is called. Most important to this study is the `verbs` attribute, which is a dictionary mapping verbs to their number of occurrences associated with a subject. A Python dictionary maps a key to a value. A fundamental constraint of the type is that keys must be

unique, while values may be duplicated. For `verbs`, the key is a string containing a verb, while the value is its number of occurrences. Due to the limitation that keys are unique, a single verb can have only one number associated with it. The verb strings used for keys were naturally the `word` attributes of verb nodes, stripped of leading or trailing dashes that occasionally appear in the treebank notation. The `do_analysis` method, in its abstracted form used by both `ProdropAnalyzer` and `NonProdropAnalyzer`, is shown in (62) below.

(62)

```
198       def analyze_tree(self, tree):
199           """
200           Analyze a single tree and return list of associated verbs found.
201           For each subject match as returned by itersubjects, update counters
202           and dictionaries accordingly.
203           A verb may appear multiple times in the returned list.
204           """
205           self.tree_count += 1
206           has_subject = False
207           valid_verbs = []
208
209           for node in self.itersubjects(tree):
210               has_subject = True
211               self.subject_count += 1
212               sibtags = []
213
214               result = self._get_associated_verb(node)
215
216               # Success. Verb found
217               if hasattr(result, 'tag'):
218                   self.subject_w_verb_count += 1
219                   update_distinct_counts(self.verb_counts, result.word)
220                   valid_verbs.append(result.word)
221
222               # Failure.
223               # Store sibling tags for reporting if no associated verb
224               # found matching allowed tags.
225               else:
226                   sibtags += result
227                   for t in sibtags:
228                       update_distinct_counts(self.ignored_tag_counts, t)
229                   self.failure_trees.add(tree)
230
231           self.tree_w_subject_count += 1 if has_subject else 0
232
233           return valid_verbs
234
235       def do_analysis(self):
236           """
237           For each tree in input_path (which may involve searching multiple
238           files if input_path is a directory), search for subject matches
239           using itersubjects and update counters and dictionaries accordingly.
240           """
241           self._reset()
242
243           print('Starting {0} search... '.format(
244               self.subject_descriptor), end=''
245           )
246           for tree in self.itertrees():
247               self.analyze_tree(tree)
248
249           print('Complete.\n')
```

The `CombinedAnalyzer` class provides an efficient way to gather data for both types of subjects. Its `do_analysis` method iterates once through the treebank data, running the `analyze_tree` method for each of the other two "analyzer" types on each tree. It provides an additional file-writing method that writes a .csv file (a simple spreadsheet format) in which each row contains a verb, the number of times it was found to be associated with a pro-dropped subject, and the number of times it was found to be associated with a non-pro-dropped subject.

A final module, *analyze_corpus*, was the entry point of the completed project. It instantiated a `CombinedAnalyzer` object and called its methods to run the analysis and write the report files.

4.5.5 Limitations. A limitation of the final data was that a verb, in the context of number of associations counted, was considered unique if it was a sequence of characters distinct from any other. The resulting data did not account for different forms of the same root word. For example, in English it could be argued that any occurrence of the past-tense "ran" should be considered an occurrence of its root verb "run" within the final analysis data. This limitation was partly necessitated out of the programmer's lack of ability to read Arabic. However, the task of accurately linking every differing form of a verb to its root verb is a large one, even using a programmer's native language. It would likely not be worth the time or potential error-prone data to address this issue.

4.5.6 Results. On a three-core 2.8GHz system, the entire program ran in under 20 seconds. The total number of trees examined was 12,628 spread among 599 .parse files. At least one pro-dropped subject was found in 6,249 trees, with 11,383 total pro-dropped subjects found. Of these pro-dropped subjects, an associated verb was identified for 9,918 of them, or 87.1%. At least one non-pro-dropped subject was found in 9,937 trees, with 27,498 total non-pro-dropped

subjects found. Of these non-pro-dropped subjects, an associated verb was identified for 23,347 of them, or 84.9%. The total number of distinct verbs found to be associated with an identified subject was 9,826. A more detailed analysis of the findings is the focus of the next chapter.

Chapter 5: Computational Analysis of Arabic Pro-Drop — Findings

5.1 Introduction

As introduced in section 4.1, the topic at hand is the analysis, using the Python programming language, of pro-drop in the OntoNotes Arabic treebank. Several scripts utilizing a custom Python library were used to run a series of experiments and collect data on Arabic pro-drop. Chapter 4 discussed the technical implementation details of both the library and these experiments. This chapter will discuss and analyze the results of each experiment.

The overarching purpose behind the experiment set was to shed light on the optionality of pro-drop in Arabic. By extracting pro-dropped subjects from a corpus and examining what surrounds them, and in what contexts they appear, perhaps some understanding about the factors that implicitly influence a speaker to use pro in Arabic will surface.

5.2 General Corpus Data

The following table provides a general overview of what was found in the corpus. This data can be seen as common to every experiment described later in this chapter. In this case, "total trees parsed" is identical to the number of the trees in the treebank. For a detailed discussion of what being an "associated verb" entails, refer back to section 4.5.3.

Table 5: General Corpus Data

12,628	**Total trees parsed**
6,249	Trees with at least 1 pro-drop
11,383	Total pro-drops in Treebank
9, 826	Total distinct associated verbs
128	Associated verbs which appear at least 30 times
33,747	Total subjects in corpus
33.73%	Percent subjects pro-dropped
66.27%	Percent subjects overt
24.37%	Mean rate of pro-drop for verbs with frequency >=30

Somewhat interestingly, at least one pro-drop occurs in nearly exactly fifty percent (49.49%) of trees in the treebank. The difficulty of identifying related verbs described in section 4.5.5 is largely responsible for the large discrepancy between the number of distinct verbs identified and the number of verbs appearing often enough to be considered statistically significant.

The overall rate of pro-drop in subjects, approximately one-third, should be noted as a baseline for comparison in the analyses discussed later in this chapter. The "Percent subjects pro-dropped" number is simply the proportion of the total number of covert subjects found to the total number of subjects found. The "Mean rate of pro-drop…" statistic is an average of pro-drop rates for individual verbs.

5.3 Verb Association Analysis

As discussed previously, some analyses involved pairing an overt or covert subject with an associated verb, i.e., the verb of the VP nested most closely to the subject. The purpose of this

was to determine if there is any correlation between Arabic pro-drop and particular verbs or verb types.

5.3.1 Psych verbs vs. action verbs. One manner of classifying verbs is deeming them "action" or "psych" verbs. An action verb describes a physical action, while a psych verb describes a mental state or event. This section seeks to provide an answer to the question of whether there is any pattern between verb type and pro-drop of a subject in a verb phrase.

The 136 verbs deemed statistically significant were classified as action, psych, or "context," a third category for verbs that can be either action or psych. An example of such a "context" verb in English is "feel." To feel an emotion is psychological, while to feel a material is a physical action. In the end, 83 verbs were classified as action, and 36 as psych. The rest were deemed too difficult to classify with certainty. The following table contains basic statistics about the verb types.

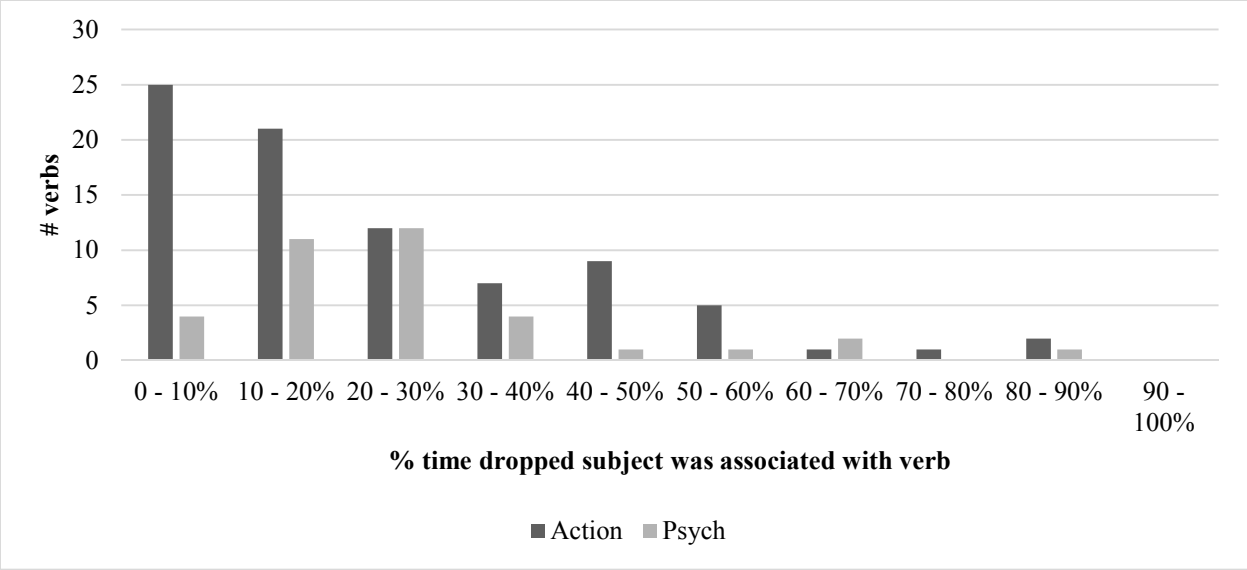
Table 6: Action/Psych Verbs and Pro-Drop

	Action Verbs	**Psych Verbs**
Pro-Drop % mean	23.75%	25.59%
Total Occurrences	5,775	1,907

The data seems to suggest that this manner of verb classification has little impact on the occurrence of pro-drop in Standard Arabic. For a given action or psych verb, the average rate of an associated subject being pro-dropped is approximately 25%, and very near the mean for all verbs.

Though the overall occurrence of pro-drop is similar for action/psych verbs as shown in Table 6, a visualatizion of the data reveals some patterns. Consider the graph below.

Table 7: Frequency of Pro-Drop with Arabic Action/Psych Verbs



The leftmost section of the graph (the 0%–10% section), for example, shows that for 25 separate action verbs, for all their appearances in the corpus, in less than 10% of these appearances the verb was associated with a pro-dropped subject. This graph reveals something interesting about the psych and action verbs. The shapes on this graph for each verb type are somewhat inverses of each other. Action verbs cluster heavily in the 0%–10% and 10%–20% sections drop off, then rise again at 40%–50%. This suggests that subjects in an action verb clause are more likely to be either very rarely pro-dropped, or that the drop occurs about half the time, depending on the verb. Psych verbs, on the other hand, look more like a traditional bell curve, with the peak in the 20%–30% range as the mean suggests. Note that no verb was associated with a pro-drop more than 90% of the time, and numbers for both types are sparse beyond 60%.

Despite the slightly different pattern in pro-drop occurrence discussed above, the data overall suggests that when classifying verbs as action or psych, these verb types have little to no impact on the occurrence of pro-drop in Standard Arabic.

5.4 Wh-Phrase Analysis

Another hypothesis tested was that the type of wh-phrase in which a subject appears might have an impact on the occurrence of pro-drop. Three different types of wh-phrases were extracted and their subjects analyzed. The types of wh-phrases found in the corpus were wh-noun phrases, wh-preposition phrases, and wh-adverb phrases.[12] The data for each is seen below.

Table 8: Wh-phrase Analysis

Tag	Total	Min. 1 Pro Subject	Pro Subjects	% Pro-drop
WHNP	10,690	1373	1404	12.8%
WHPP	47	20	20	42.6%
WHADVP	600	196	201	32.7%

In the table, "total" is the number of wh-phrases in the corpus found of each type. The next column is the number of phrases of each type found to contain at least one pro-dropped subject. The fourth column is the total number of pro subjects found in each wh-phrase type, and the final column shows the percentage of wh-phrases of each type that contained at least one pro subject. The use of pro-drop seems to vary significantly between different types of wh-phrases.

Table 8 shows that the frequency of pro-drop varies significantly between different forms of wh-phrases. Pro is relatively rare in wh-noun phrases relative to the overall mean of approximately 25%, and more common in wh-preposition phrases, although far fewer examples

[12] An additional tag, WHADJP, had no occurrences in the corpus and is omitted from Table 8.

of this type were found in the corpus. The final type—wh-adverb phrases—is slightly above the mean.

5.5 Conclusion

This chapter discussed several examples of analyses that could be performed to examine what factors influence the optionality of Arabic pro-drop. When classifying verbs as action or psych, it was discovered that pro subjects occur with approximately the same frequency regardless the type of verb they are associated with. Data from a different analysis suggested that pro in wh-noun phrases is relatively rare, while it may be comparatively frequent in wh-preposition phrases. General data about the corpus provided a macro overview of the occurrence of Arabic pro-drop.

Chapter 6: Conclusion

6.1 Summary

This work has discussed pro-drop in Arabic in several contexts. Chapter 2 investigated five properties considered crucial to formulation of the null-subject parameter. These properties were discussed in a generalized manner by comparing null-subject and non-null-subject languages. In particular, Standard Arabic was investigated with regard to each property. The language was determined to correlate with other NSLs in its use of missing subjects, free inversion, and null resumptive pronouns. The fourth property, violation of the that-trace filter, was shown to be in an interesting state: Standard Arabic can *seem* to obey the that-trace filter as seen in non-null subject languages, but cannot be said to do so. Its seeming obedience to the filter actually comes from an independent constraint of the Arabic complementizer *ʔanna*. The final property, long-wh movement of subjects, provides a puzzling result. Arabic patterns clearly align with NNSLs such as English in its disallowance of this sort of movement. There is no clear explanation for this phenomenon at this time.

Chapter 3 discussed the Saudi Najdi dialect of Arabic with regard to the aforementioned null-subject parameter properties. A new dataset was presented for this dialect. It was shown to closely parallel Standard Arabic; the sole syntactic difference was the lack of the mood assigner *ʔan* in Najdi. In a broader sense, Chapter 3 illustrated how dialects of a language can show significant differences in syntax from their root languages.

Chapters 4 and 5 discussed using the Python language to gather metadata about Arabic pro-drop from a large Standard Arabic corpus. This data was used in discussing the frequency of occurrence of pro-drop in certain contexts, and the optionality of pro-drop.

6.2 Future Research

Though this work has attempted to clarify several contentious issues regarding the null-subject parameter of Standard and Najdi Arabic, an important question remains. Arabic is a puzzling language in that it exhibits four of five expected properties, but clearly parallels NNSLs with regard to long-wh movement of subjects. Future work could attempt to provide an explanation of this phenomenon.

The Python tool used in the analyses of Chapters 4 and 5 could provide avenues of future research. Any corpus notated in the Penn Treebank manner could be parsed by this tool. Other Arabic corpora could be analyzed in a similar manner to expand the scope of the data on Arabic pro-drop, or corpora of other languages could be examined. An interesting topic may be comparing the results of the 2003 corpus data used in this work to earlier or later corpora to examine language change in Arabic. A worthy question is to what degree the occurrence of pro-drop in Arabic is changing over time.

References

Akkal, A. (1996). How SVO is SVO in Standard Arabic. In A. Fassi-Fehri (Ed.), *Liguistique comparee et langues au Maroc* (pp. 101–127). Rabat: Universite Mohamed V: Publications de la Faculte des Lettres et des Sciences Humaines.

AlAlamat, H. (2014). Pro and Verb Movement in Arabic Syntax. *European scientific journal*, *10*(2).

Alotaibi, M., & Borsley, R. D. (2013). Gaps and resumptive pronouns in Modern Standard Arabic. In Stefan Muller (Ed), *Proceedings of the 20th International Conference on Head-Driven Phrase Structure Grammar* (p.6).

Al-Sweel, A. I. (1981). *The verbal system of Najdi Arabic: A morphological and phonological study.* (Doctoral dissertation). University of Washington.

Aoun, J. (1981). ECP move α, and subjacency. *Linguistic Inquiry, 12*(4), 637–645.

Bayer, J., & Salzmann, M. (2013). That-trace effects and resumption–How Improper Movement can be repaired. *Repairs: The Added Value of Being Wrong*, *27*, 275.

Berjaoui, N. (2009). *The Empty Category Principle in English and Standard Arabic*. Lincom GmbH.

Bies, A. & Maamouri, M. (2003). Penn Arabic Treebank Guidelines. Linguistic Data Consortium, Philadelphia, PA.

Camacho, J. (2013). *Null subjects* (Vol. 137). Cambridge University Press.

Chomsky, N. (1981). *Lectures on government and binding: The Pisa Lectures*. Dordrecht: Foris Publications.

Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding.* Linguistic Inquiry Monograph, Vol. 6. Cambridge, Massachusetts: MIT Press.

Chomsky, N. (1986a). *Barriers.* Linguistic Inquiry Monograph, Vol. 13. Cambridge, MA.: MIT Press.

Chung, S., (1998). *The design of agreement: Evidence from Chamorro*. Chicago: University of Chicago.

Cognola, F. (2013*). Syntactic variation and verb second : A German dialect in Northern Italy*. Amsterdam: John Benjamins Publishing Company. Doctoral dissertation, University of Southern California.

Dubost, K. (2008, May 6). UTF-8 Growth on the Web. Retrieved October 8, 2014.

Ennaji, M. (1991). Subject Clitics in Modern Standard Arabic, Revised version. Ms, Faculty of letters, Fes.

Hahn, Michael (n.d.). Pronominal Null Conjuncts in Arabic. Retrieved from https://depts.washington.edu/hpsg2011/pp/Hahn.pdf.

Jlassi, M. (n.d.). On the existence of subject clitics in Arabic: Evidence from particle clitics in Tunisian Arabic. Retrieved from http://www.hf.uio.no/ilos/forskning/aktuelt/arrangementer/konferanser/2011/challenging-clitics/program/Abstract%20Jlassi.pdf

Kayne, R. (1984*). Connectedness and binary branching*. Dordrecht: Foris Publications.

Kayne, R.S. (1975). *French syntax: The transformational cycle*. Cambridge: MIT Press.

Kenstowicz, M. (1989). The null subject parameter in modern Arabic dialects. In Jaeggli, O. and Safir, J. (Eds), *The null subject parameter* (pp. 263–275). Dordrecht: Kluwer Academic Publishers.

Lewis, M. Paul, Gary F. Simons, & Charles D. Fennig (eds.). (2014). *Ethnologue: Languages of the world, Seventeenth edition*. Dallas, Texas: SIL International. Online version: http://www.ethnologue.com.

Lewis, R. E., Jr. (2013). Complementizer agreement in Najdi Arabic (Order No. 1544950). Available from ProQuest Dissertations & Theses Global. (1441080934).

Maamouri, Mohamed et al. (2008). Arabic Treebank part 3 Readme. Linguistic Data Consortium.

Modesto, M. (2000). Null subject without 'rich' agreement. In Kato & Negrão (2000), 147–174.

Montalbetti, Mario. 1984. *After Binding*. Ph.D. thesis. MIT.

Neeleman, A., & Szendrői, K. (2005). Radical pro drop and the morphology of pronouns. *Linguistic Inquiry, 38*(4), 671–714.

Nicolis, M. (2008). The null subject parameter and correlating properties. The limits of syntactic variation, 132, 271.

Ordóñez, F. & Olarrea, A. (2006). Microvariation in Caribbean/non Caribbean Spanish interrogatives. *Probus, 18*(1), pp. 1–158. Retrieved 10 Aug. 2014, from doi:10.1515/PROBUS.2006.003

Ouhalla, J. (1997). Remarks on focus in Standard Arabic. In M. Eid and R. R. Ratcliffe (Eds.), *Perspectives on Arabic Linguistics X: Papers from the Tenth Annual Symposium on Arabic Linguistics* (pp.9–45). Amesterdam/Philadelphia: John Benjamins.

Perlmutter, D. (1971). *Deep and surface structure constrains in syntax*. New York: Holt, Rinehard and Winston.

Peters, T. (2004). PEP 20 -- The Zen of Python. Retrieved October 8, 2014.

Riemsdijk, H. van and Williams, E., 1986. *Introduction to the Theory of Grammar*. Cambridge, MA: MIT Press.

Rizzi, L. (1980). *Negation, Wh-movement and the Null-Subject Parameter*. Ms. Coshuza, Italy: Universita Della Callabria.

Rizzi, L. (1982). *Issues in Italian syntax*. Dordrecht: Foris Publications.

Rizzi, L. (1986). Null objects in Italian and the theory of pro. *Linguistic Inquiry, 17,* 501–57.

Rothman, J., & Iverson, M. (2007). The syntax of null subjects in L2 Spanish: Comparing two L2 populations under different exposure. *Revista española de lingüística aplicada*, (20), 185–216.

Safir, K. (1985). *Syntactic chains*. London: Cambridge.

Sells, P., 1984. *Syntax and semantics of resumptive pronouns*.(Unpublished PhD Dissertation). Amherst: University of Massachusetts.

Unicode Consortium, The. (2014). *The Unicode Standard*, Version 7.0.0, 67.

Van Rossum, G. (2009, February 14). What's new in Python 3.0. Retrieved October 8, 2014.

Weischedel, Ralph, et al. (2013). OntoNotes Release 5.0 LDC2013T19. Philadelphia: Linguistic Data Consortium.