Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2006-07-13

Crossing Dependencies in Persian

Jonathan M. Dehdari
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Linguistics Commons

BYU ScholarsArchive Citation
Dehdari, Jonathan M., "Crossing Dependencies in Persian" (2006). *Theses and Dissertations*. 732.
https://scholarsarchive.byu.edu/etd/732

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

CROSSING DEPENDENCIES IN PERSIAN

by

Jonathan Dehdari

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Arts

Department of Linguistics & English Language

Brigham Young University

August 2006

Copyright © 2006 Jonathan Dehdari

Typeset in Palatino using $\LaTeX\,2_\varepsilon$

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jonathan Dehdari

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Deryle Lonsdale, Chair
Date	Alan Manning
Date	Mark Davies

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jonathan Dehdari in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Deryle Lonsdale
Chair, Graduate Committee

Accepted for the Department

John S. Robertson
Associate Chair, Department of Linguistics
and English Language

Accepted for the College

Gregory Clark
Associate Dean, College of Humanities

ABSTRACT

CROSSING DEPENDENCIES IN PERSIAN

Jonathan Dehdari

Department of Linguistics & English Language

Master of Arts

Languages occasionally have syntactic constructions that are difficult, if not impossible, to describe using a context-free grammar. One such construction is a crossing dependency. Crossing dependencies have been well studied for Dutch and Swiss German (Huybregts, 1976; Shieber, 1985), and recently for Tagalog (Maclachlan and Rambow, 2003). In this paper I propose that Persian exhibits crossing dependencies.

In this SOV language, a light verb construction in the future tense becomes interrupted by a future auxiliary verb, which agrees with its subject in person and number. The future auxiliary also splits passive constructions in a similar manner. These forms present interesting challenges for computational models of language. I will discuss implications of this phenomenon within current formal and linguistic theories.

ACKNOWLEDGEMENTS

I am greatly indebted to Deryle Lonsdale for advising me these past five years. This thesis would not have been possible without his years of guidance and patience. I am also grateful for the kindness and support of John Robertson, Alan Manning, Mark Davies, Phyllis Daniel, and the entire Linguistics Department. I'd like to acknowledge the financial support of the Office of Research and Creative Activities, the Linguistics Department, and BYU.

I am very grateful for Mary and Hooshang Farahnakian, the Center for Language Studies, and Lloyd Miller for helping me better understand the Persian language, music, and culture.

I'd like to thank my family for their love and support throughout my life. I am also grateful to the Clement family for their warmth and hospitality. I would like to extend my warmest gratitude to Carolyn Dehdari for her support, interest, and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The aim of this thesis is to show crossing dependencies in Persian and possible ways to theoretically and computationally account for them. In a crossing dependency, a language includes a sequence of words like $a_1\ b_2\ c_1\ d_2$. Words a and c exhibit some type of dependency, such as case, ϕ-feature, or compound lexeme. However, words b and d exhibit another dependency, resulting in crossing dependencies. In Persian a light verb construction becomes interrupted by a future auxiliary verb, which itself agrees with its subject in person and number. For example, in the sentence *ānhā*$_1$ *dæst*$_2$ *xāhænd*$_1$ *zæd*$_2$ 'they will clap', the light verb construction *dæst zæd* 'clap' becomes split by the future auxiliary *xāh-ænd* 'will-3P', which agrees with the subject *ānhā* 'they' in person and number.

Crossing dependencies are interesting linguistically and computationally, as they can present challenges for restricted theories of syntax. Since their structures are not context-free, traditional methods of describing the relations between words are not fully adequate. This is because the only way derive such constructions is through incorrect linguistic derivations of their context-free form—if they can be derived at all. Some crossing dependencies are types of scrambling, common in freer-word-order languages like German, Czech, or Warlpiri. Other crossing dependencies are fixed-word-order, required, and bounded as to the number of crossing dependencies. Others are optional, but have no theoretical limit on their number of crossings. This paper will treat the second and third types. See Karimi (2003) for a thorough treatment of the first type.

The outline of this work is as follows. First I will introduce crossing dependencies and discuss their context within linguistics and computer science. Then I will review how they have been explained in various formal and linguistic theories. After a brief survey of the Persian language and its syntax, I will present structures in Persian that display crossing dependencies. I will show that the same mechanisms used to account for crossing dependencies in previously described languages also aptly account for the Persian structures. A comparison between the Persian forms and those of other languages is also put forward, as well as concluding remarks.

Chapter 2

Background and Literature Review

An overview of formal languages is given in this chapter, and their relation to natural languages. This initial information will serve as a context for the syntactic analyses in Chapter 4. To start, some terms will be initially defined.

2.1 Initial Definitions

grammar – a device to generate and recognize all forms and only the forms of a given language.

language – either a natural language or a formal language.

strong generative capacity (SGC) – the set of structures that can be generated by a grammar.

Turing machine – a hypothesized machine that is theoretically equivalent to any computer in what it can and cannot perform.

weak generative capacity (WGC) – the set of strings that can be generated by a grammar.

2.2 Review of Literature and Concepts

With the advent of computers, attempts were made to mechanistically describe what natural language is and what it is not. Chomsky (1956) helped lay the foundations for describing languages generatively. A hierarchy, currently known

Type	Language	Grammar
0	recursively enumerable	Turing machine
1	context-sensitive	context-sensitive
2	context-free	context-free
3	regular	finite-state automata

Table 2.1: The Chomsky hierarchy

as the Chomsky hierarchy of languages, classified formal languages in increasing order of restrictiveness, as seen in Table 2.1. The least restricted are *recursively enumerable* (type 0), which are languages described by a Turing Machine[1]. A grammar production for these languages can look like $aAbB \rightarrow aCb$. The lower-case letters represent primitive symbols known as terminals, capital letters represent variables called non-terminals, and the rightward arrow represents a string rewriting from the left-hand side to the right-hand side (in the case of generation). The left-hand side of recursively enumerable productions must be non-empty. Any algorithm (in the modern sense) has the weak generative capacity to describe any recursively enumerable (RE) language. For example, a RE language can allow for sentences that grow exponentially, rather than incrementally. The computation of RE languages can be extremely difficult.

Context-sensitive languages (type 1) are a proper subset of recursively enumerable languages, being equivalent to linear-bounded non-deterministic automata. Productions, or grammatical rules for a context-sensitive language can look like $aAbBc \rightarrow aCdDbBc$. The right-hand side of a production must be equal to or larger than the left-hand side (Stabler, 2004), thus the right-hand side of the production must be non-empty. Linguists might be more familiar with another notation for context-sensitive grammars: $A \rightarrow CdD \ / \ a\_\_ \ bBc$. Savitch (1987) discusses

[1] or the λ-calculus (Turing, 1937; Barendregt, 1997), Markov algorithms (Markov, 1960), among others

why context-sensitive grammars are a poor choice for treating natural language. He asserts that "all the recursively enumerable languages can be found among the context-sensitive languages" (pg. 362). Type 1 languages are not as widely studied in mathematical linguistics or computer science as other language types.

Since the original hierarchy in Chomsky (1959), there have been other formal languages found to be proper subsets of context-sensitive languages and proper supersets of context-free languages (Aho, 1968). *Indexed languages* can be described by a context-free grammar with a stack (Gazdar and Mellish, 1989). A production for such a language could look like $A[i] \rightarrow BaC$, where non-terminals inherit their parent node's stack and optionally pop one element off. Linguistically, indexed grammars allow features to dynamically appear (push) and get checked-off (pop) without hard-coding them into the grammar. Gazdar and Pullum (1985) assert that all grammatical natural language phenomena can be handled by indexed grammars.

Mildly context-sensitive languages are a proper subset of indexed languages and a proper superset of context-free languages. A production for these languages can look like those for indexed languages, but only one non-terminal may inherit its parent node's stack (Vijay-Shanker, 1987). Their computational complexity is considerably lower than the languages previously mentioned. Tree adjoining grammars (Joshi *et al.*, 1975), head grammars (Pollard, 1984), and combinatory categorial grammars (Steedman, 1985) generate mildly context-sensitive languages.

Context-free languages (type 2) can be described by a grammar having a production like $A \rightarrow BaC$, where the left-hand side has only one non-terminal node. They are sometimes referred to as PDA languages, since they can be described by pushdown automata, which are finite-state automata with a stack (Kracht, 2003, pg. 117). Categorial grammars have the weak generative capacity of a context-free grammar (CFG). CFGs are widely used in probabilistic and symbolic parsers of natural languages and programming languages.

Deterministic context-free languages, or deterministic PDA languages, are a proper subset of context-free languages and a proper superset of regular languages. A grammar for these languages is unambiguous (Hopcroft and Ullman, 1979, pg. 255). They are rarely studied in linguistics, although they offer potential relevance in the area of sentence processing, since backtracking is not permitted in parsing these languages.

Regular languages (type 3) have grammars which must only have one node on the left-hand side, and either a terminal node or a non-terminal node and a terminal node on the right-hand side (Vogel *et al.*, 1996). A regular grammar can have a production like $A \rightarrow Ba$. An alternative notation is expressed in the form of *regular expressions*[2], although the current use of the term extends to non-regular grammars. Regular grammars are also often expressed in the form of finite-state automata. Since regular languages are so restricted, they can be parsed extremely efficiently.

Our knowledge of formal languages has increased considerably since the original descriptions in the 1950's and 1960's. Table 2.2 shows an expanded hierarchy of formal languages, adapted from Hopcroft and Ullman (1979); Partee *et al.* (1990); Sipser (1997).

Given this hierarchy of language types, attempts were made to determine where the syntax of natural language fits. Chomsky stated "we should like to accept the least 'powerful' theory that is empirically adequate" (1965, pg. 62). He asserted that natural language could not completely be described using a regular grammar (1956, pg. 115). Essentially, recursive center clausal embedding is not regular:

(1) a. If S_1, then S_2.
    b. Either S_3, or S_4.
    c. The man who said that S_5, is arriving today.

[2]See Friedl and Oram (2002) for an in-depth treatment.

Language
Non-Turing-acceptable
Recursively enumerable
Recursive/ Decidable
Context-sensitive
Indexed
Mildly context-sensitive
Context-free
Deterministic context-free
Regular
Finite

Table 2.2: An extended hierarchy of formal languages

He then addressed context-free grammars, stating that they have "no place for discontinuous elements" (pg. 120). He argued that English constructions like the one found in (2) were not context-free (1956, pg. 120):

(2) the man had be-en tak-ing the book.

The past perfect *had* requires a past participle to follow it. This is usually manifest as the suffix *-ed/-en*. Also, the progressive *be* requires a present participle afterward, which is indicated by the suffix *-ing*. These morphemic dependencies cross each other. Chomsky viewed the above example as a discontinuous morphophonemic derivation from originally continuous morphemes. Accordingly he adopted a more powerful mechanism of transformations, based on the work of Zellig Harris (1952). Crossing dependencies like the one above have been observed for other languages as well, although there is no treatment of crossing dependencies for Persian to the author's knowledge.

Over the next 25 years many others presented arguments in favor of the non-context-freeness of natural language, including: Bar-Hillel and Shamir (1960) who based their assertion on an English construction involving *respectively*, Postal (1964) on Mohawk incorporation, Bach (1974) on English number agreement, Huybregts (1976) on Dutch cross-serial dependencies, and Bresnan (1978) on *wh*-extraction and number agreement.

Of interest to this paper is the case of Dutch cross-serial dependencies (CSDs). Cross-serial dependencies are a type of crossing dependency where any number of crossings are chained together. In Dutch complement clauses, complementizing finite verbs allow their complement clause to circumscribe them, with noun phrases preceding the complementizing verb and non-finite verbs following it:

(3) . . . dat Jan de kinderen zag zwemmen
. . . that Jan the children saw swim

'. . . that Jan saw the children swim'

Subsequent complement clauses may follow this same pattern around each complementizing verb with no upper bound on the number of times this may occur. Bresnan *et al.* (1982) offered an LFG account for these structures, making use of non-endocentric c-structures.

The notion of natural language not being context-free was fairly uncontested until Pullum and Gazdar (1982) showed that the stringsets of each of these phenomena could in fact be explained with a CFG, and some even with a regular grammar. Their claim was not that natural language was necessarily context-free, but rather that "every published argument purporting to demonstrate the non-context-freeness of some natural language is invalid, either formally or empirically or both" (§ 7).

A prominent argument came soon after when Huybregts (1984) and Shieber (1985) declared that Swiss German cross-serial dependencies could not be weakly generated by a CFG. Their form was very similar to the Dutch counterpart, but with one crucial difference: overt case agreement between each verb and its corresponding noun. That is, the noun must either be inflected for the accusative or the dative case, depending on which verb is used:

(4)
...das	mer	d'chind	em Hans	es huus	lönd	hälfe	aastriiche
...that	we	the children-ACC	Hans-DAT	house-ACC	let	help	paint

'...that we let the children help Hans paint the house'

The renowned finding was not without criticism, however. Manaster Ramer (1988) argued that Shieber (1985) contains a flaw in mathematical reasoning, stating that the paper assumes that "the number of verbs that govern the dative must equal the number of actual NPs in the dative case in the sentence, and likewise for the accusative" (pg. 101). Instead he offers an alternative where the number of dative (or accusative) NPs are no greater than the number of dative-governing verbs. At any rate Gazdar and Pullum (1985) emphasize that regardless of whether natural language (NL) as a whole is context-free or not, "the overwhelming majority of the structures of any NL can be elegantly and efficiently parsed using context-free parsing technologies."

Since the work on Dutch and Swiss German—both West Germanic languages—little research has been conducted on cross-serial dependencies in other languages. A notable exception is Maclachlan and Rambow (2003) on Tagalog, an Austronesian language. Unlike the Dutch and Swiss German CSDs, the verbs precede the NPs in Tagalog CSDs:

(5)
Nagisip	na	bumili	si Pedro	ng bulaklak
AT-thought	LK	AT-buy	NOM-Pedro	flower

'Pedro thought to buy (of buying) a flower.'

While such orderings are optional, they nevertheless are grammatical and thus need to be accounted for in some way. Certainly the Dutch (3), Swiss German (4), and Tagalog (5) examples exceed the strong generative capacity of a context-free grammar, as well as the English example (2) from a morphemic perspective.

Having reviewed crossing dependencies in English, Dutch, Swiss German, and Tagalog, I will discuss the Persian language in the next chapter. Specifically, two constructions of the language are analyzed—the light verb construction and the passive construction.

Chapter 3

Persian

The Persian language, or Farsi, is an Indo-European language natively spoken by about 60 million people in Iran, Afghanistan, Tajikistan, and surrounding areas. The language has remained remarkably stable since the eighth century. It has a subject-object-verb word order, but has some head-initial structures. This paper will focus on the written form of modern Iranian Persian. An explanation of the orthographic conventions used in this work is found in Appendix C.

3.1 Persian Light Verb Constructions

In addition to normal verbal forms, Persian makes extensive and highly productive use of Light Verb Constructions (LVC)[1]. Such formations are composed of a light verb (LV) and a non-verbal element (NV). The non-verbal element can be a noun, adjective, preposition, prepositional phrase, or idiomatic structure, as seen in (1) from Megerdoomian (2002).

(1) a. fæks kærdæn
fax do
'to fax'

b. delxor kærdæn
annoyed do
'to annoy'

[1]Other works also refer to Persian LVCs as Complex Predicates, Complex Verbs, and Compound Verbs.

c. æz beyn bordæn
from between take
'to destroy'

d. piš ræftæn
before go
'to advance'

e. dæst be dæst kærdæn
hand to hand do
'to hesitate'

These constructions can range from highly compositional, like (1a), to highly non-compositional, like (1c). The more compositional LVCs tend to contain non-verbal elements that are nouns or adjectives[2], and light verbs that are frequently occurring. Conversely, the more non-compositional LVCs tend to contain non-verbal elements that are native, concrete, or more complex than a simple noun, and light verbs that are less frequent. To illustrate the number of LVs in Persian, I have compiled a sorted listing of 35 light verbs in Table 3.1. It was produced using pattern matching tools on Dr. Amir Shakib-Manesh's digital Persian-English dictionary. Each entry was verified in multiple corpora and other dictionaries[3].

Most nouns and many adjectives combine with a light verb to form an LVC. The number of non-verbal elements has no upper bound, as the number of loan-words and neologisms continues to grow. Some examples include:

(2) a. ček kærdæn
check do
'to check'[4]

b. imeyl zædæn
email hit
'to email'

[2]They are also usually borrowed or abstract.

[3]The corpora primarily consist of a 20 million word Kayhan online news text and a 10 million word BBC Persian online news text.

[4]Interestingly the English word *check* ultimately derives from the Persian word *šāh* 'shah, king'.

Light verb	Gloss	Example	Literal Trans.	Gloss
kærdæn	‘do’	*peydā kærdæn*	visible do	‘find’
šodæn	‘become’	*vāred šodæn*	arriving become	‘enter’
dādæn	‘give’	*pošt dādæn*	back give	‘lean’
zædæn	‘hit’	*dæst zædæn*	hand hit	‘clap’
budæn	‘be’	*šāmel budæn*	containing be	‘include’
dāštæn	‘have’	*dust dāštæn*	friend have	‘like’
gereftæn	‘take’	*tæ’ælloq gereftæn*	attachment take	‘accrue’
ræftæn	‘go’	*qærāvol ræftæn*	sentinel go	‘aim’
kešidæn	‘pull’	*dæst kešidæn*	hand pull	‘desist’
ændāxtæn	‘throw’	*dæst ændāxtæn*	hand throw	‘spoof’
xordæn	‘eat’	*šekæst xordæn*	break eat	‘lose’
gozāštæn	‘put’	*qāl gozāštæn*	smelting put	‘leave waiting’
āværdæn	‘bring’	*’æml āværdæn*	act bring	‘manufacture’
sāxtæn	‘build’	*mærbut sāxtæn*	related build	‘affiliate’
goftæn	‘say’	*mæhræmāne goftæn*	confidential say	‘confide’
yāftæn	‘find’	*dæst yāftæn*	hand find	‘attain’
bæstæn	‘close’	*čæšm bæstæn*	eye close	‘blindfold’
bordæn	‘carry’	*nām bordæn*	name carry	‘mention’
āmædæn	‘come’	*’æml āmædæn*	act come	‘ripen’
rixtæn	‘pour’	*foru rixtæn*	downward pour	‘collapse’
oftādæn	‘fall’	*eteffāq oftādæn*	event fall	‘happen’
næmudæn	‘appear’	*xonsā næmudæn*	neutral appear	‘annihilate’
resāndæn	‘extend’	*ziān resāndæn*	loss extend	‘injure’
bærdāštæn	‘pick up’	*dæst bærdāštæn*	hand pick up	‘desist’
jostæn	‘search’	*del jostæn*	heart search	‘be agreeable’
bæxšidæn	‘forgive/bestow’	*ruhiye bæxšidæn*	morale bestow	‘uplift’
xāndæn	‘read/sing’	*færā xāndæn*	back read	‘summon’
peydā kærdæn	‘find’	*šib peydā kærdæn*	slope find	‘decline’
residæn	‘arrive’	*xedmæt residæn*	service arrive	‘wait upon’
didæn	‘see’	*āsib didæn*	injury see	‘sustain an injury’
bāxtæn	‘lose/play’	*jān bāxtæn*	soul lose	‘self-sacrifice’
kubidæn	‘pound’	*xāl kubidæn*	spot pound	‘tattoo’
gærdidæn	‘turn’	*montej gærdidæn*	result turn	‘accrue’
čidæn	‘clip/pluck’	*čæšm čidæn*	eye pluck	‘counteract’
boridæn	‘cut’	*omid boridæn*	hope cut	‘despair’

Table 3.1: Some Persian light verbs in decreasing frequency

c. čæt kærdæn
chat do
'to chat (online)'

Nouns may also appear in prepositional phrases that serve as a non-verbal element. Nominal compounding also indicates their non-finiteness.

Non-verbal elements combine with light verbs in idiosyncratic distributions. A common way of encoding their combinatory possibilities in current dictionaries is to store possible light verb usage in the lexicon entry of each non-verbal element.

3.2 Split Light Verb Constructions

Unlike most other languages with light verb constructions, Persian LVCs may become split, by accusative pronominal clitics, noun/determiner phrases, prepositional phrases, or certain auxiliary verbs:

(3) a. ānhā dæst=æš zæd-ænd
they hand=3S.ACC hit-3P
'They touched it.'

b. xāb-e bæčče rā did-æm
dream-GEN child ACC saw-1S
'I dreamt of the kid.' (adapted from Megerdoomian (2002))

c. ānhā dæst be tæzāhor-āt zæd-ænd
they hand to demonstration-PL hit-3P
'They embarked on demonstrations.'

d. ānhā dæst xāh-ænd zæd
they hand will-3P hit
'They will clap.'

e. ānhā dæst be tæzāhor-āt xāh-ænd zæd
they hand to demonstration-PL will-3P hit
'They will embark on demonstrations.'

In (3d) not only is the LVC interrupted by the future auxiliary *xāhænd*, but this interrupting word contains information, or features, that must agree with the subject: number and person. Failure to achieve agreement in both pairs results in either full or partial ungrammaticality.

(4) a. *ānhā peydā xāh-æm kærd
they visible FUT-1S did
'They/I will find.'

b. *ānhā ketāb xāh-ænd ræft
they book FUT-3P went
'* They will book-go.'

Lack of ϕ-feature agreement in (4a) clearly indicates ungrammaticality, rather than semantic unacceptability. Sentences which have intransitive (light) verbs are always ungrammatical unless the NV lexically corresponds to the LV. What's more, it is apparent that (4b) is a syntactic issue since the semantic class of word(s) occupying the NV/bare noun slot generally gives no indication as to whether the sentence will be felicitous or not[5].

3.3 Split Passive Constructions

The future auxiliary also interrupts passive constructions. Normal passive sentences are formed by suffixing the past participle morpheme *e* to the past-tense verb stem, then using a normally inflected form of the verb *šodæn* 'to become'. Light verb constructions are passivized in a similar way, attaching the past participle morpheme to the light verb:

(5) a. ānhā gošud-e šod-ænd
they open-PSPT became-3P
'They were opened.'

b. ānhā be zæmin zæd-e šod-ænd
they to earth hit-PSPT became-3P
'They were overthrown.'

[5]This is particularly the case for sentences using (light) verbs like *kærdæn* 'do' or *ræftæn* 'go', for example. Other sentences can be more dependent on the semantic class of the NV/bare noun, such as those using transitive (light) verbs like *xordæn* 'eat' or *zædæn* 'hit'. Like many questions involving the nature of Persian LVCs, they exhibit dualistic properties that greatly vary depending on the specific light verb involved, and to a lesser extent, the non-verbal element.

If the sentence is in the future tense, however, the future auxiliary separates these two words. The subject and the future auxiliary share person and number information, or features, while the past participle verb and the passive verb share a passive feature. This results in crossing dependencies:

(6) a. ānhā gošud-e xāh-ænd šod
they open-PSPT FUT-3P became
'They will be opened.'

b. ānhā be zæmin zæd-e xāh-ænd šod
they to earth hit-PSPT FUT-3P became
'They will be overthrown.'

Failure of either the past participle verb in the passive construction to have proper passivization markers (i.e. *verb*-e + šodæn) results in ungrammaticality, as does failure of the subject and future auxiliary to agree in person and number features:

(7) a. *ānhā gošud xāh-ænd šod
they open FUT-3P became
'They will be open.'

b. *ānhā gošud-e xāh-ænd ræft
they open-PSPT FUT-3P went
'* They will went opened.'

c. *ānhā gošud-e xāh-æm šod
they open-PSPT FUT-1S became
'They will be opened.'

Like (4), these examples represent a crossing of information at a syntactic level. Thus substituting other words and/or morphemes that fail to meet the requirements mentioned above will predictably result in ungrammaticality.

3.4 Comparison

Crossing dependencies in Persian split light verb constructions and split passives exhibit some similarities, although split LVCs are more complex overall. Whether they are split or not, both constructions maintain their compositionality or lack thereof. Passivization can syntactically involve any verb, as long as the verb is in the form of a past participle. On the other hand, light verb constructions occur idiosyncratically.

Both constructions are split by the future auxiliary *xāstæn*. As Persian is SOV, we would normally expect to find this word at the end of the sentence. But this is not the case: the future auxiliary almost always occurs as the second-to-last word in a complement clause[6]. This could indicate a split headedness in Persian, which will be discussed further in section 4.2.

[6] Elliptical forms are naturally an exception.

Chapter 4

Structural Analyses

This chapter will synthesize the findings of the previous two chapters to offer multiple explanations of the phenomena mentioned in Chapter 3. The split light verb constructions and split passive constructions both result in crossing dependencies when a future auxiliary separates their two parts[1]. These phenomena can be represented visually as a line connecting the subject and the future auxiliary, and another line connecting either the non-verbal element with the light verb (1a) to (1b) , or the past participle with the passive verb (1c):

(1) a. ānhā dæst xāh-ænd zæd
they hand FUT-3P hit
'They will clap.'

b. ānhā dæst be tæzāhor-āt xāh-ænd zæd
they hand to demonstration-PL FUT-3P hit
'They will embark on demonstrations.'

c. ānhā gošud-e xāh-ænd šod
they open-PSPT FUT-3P became
'They will be opened.'

[1] I first observed these crossing dependencies while implementing a syntactic parser for Persian (Dehdari and Lonsdale, 2007, forthcoming).

$$
\begin{array}{lcl}
\mathtt{S} & \rightarrow & \mathtt{NP\ VP} \\
\mathtt{NP} & \rightarrow & \mathtt{N} \\
\mathtt{VP} & \rightarrow & \mathtt{NV\ V'} \\
\mathtt{V'} & \rightarrow & \mathtt{V\ LV}
\end{array}
$$

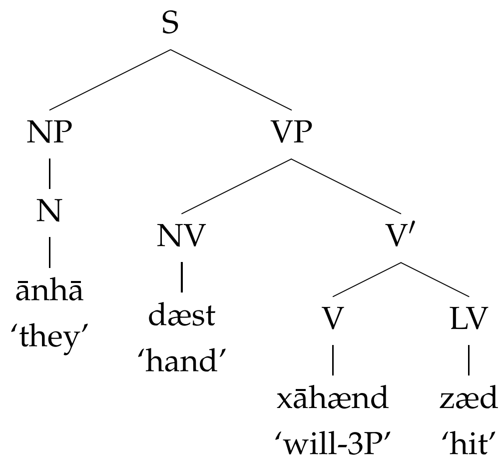


Figure 4.1: A basic CFG and derivation for LVC crossing dependencies

4.1 Context-free Grammar

As the crossing dependencies in both the Persian LVC and passive types seem to have an upper bound in length, they can be weakly described by a context-free grammar. Clearly a CFG would fail to appropriately capture the structural relations of both types, hence these forms are not strongly context-free. A basic context-free grammar to capture strings of Persian crossing dependencies could look like the left-hand side of Figure 4.1. This grammar is mostly written in Chomsky normal form, omitting terminal productions. The resulting derivation would look like the right-hand side of Figure 4.1. The node labels may be renamed to suit one's theoretical persuasion.

The important point to note with this particular grammar is that it recognizes many ungrammatical strings. For example, subjects which do not agree with the future auxiliary are accepted. Also, non-verbal element/light verb pairings for which no lexical entry exists are likewise accepted as grammatical. This situation normally would not be problematic for a grammar acting only as a string acceptor, were it not for the intransitive verbs which function as light verbs. That is, even if an NV/LV pairing was not found in the lexicon, it would still be grammatical—it would be interpreted as an bare indefinite/non-specific object and a normal transitive verb, in the case of a noun in the NV position. On the other hand, light verbs which also serve as normal intransitive verbs, such as *ræftæn* 'go', *āmædæn* 'come', *residæn* 'arrive', would render non-lexical pairings as ungrammatical, as was shown in (7b) of the previous chapter and repeated here:

(2) a. *ānhā gošud-e xāh-ænd ræft
they open-PSPT FUT-3P went
'* They will went opened.'

Feature requirements can overcome the problem of overgeneration in the grammar found in Figure 4.1. An easy way to implement featural restrictions in a context-free grammar is to simply create different nodes for all possible combinations of features. For a grammar with relatively few features, this is a trivial task. However, when the number of feature combinations is large, this approach becomes increasingly impractical. The number of possible different non-terminal nodes at a given position is the Cartesian product of all feature sets underneath it that have not been yet resolved locally:

$$X_1 \times X_2 \times \ldots \times X_n = \{\, (x_1, x_2, \ldots, x_n) \mid x_1 \in X_1 \wedge x_2 \in X_2 \wedge \ldots \wedge x_n \in X_n \}$$

An abbreviated example of a grammar which makes use of such featural alternations is seen in Figure 4.2. Given the crossing feature sets, this grammar swells to 210 alternations at the V′ node, in order to accurately accept a string of just four words. A complete implementation of this grammar in Perl 6 was written for this thesis and is found in Appendix A[2]. As an acceptor, the program will output "Grammatical" if the input sentence is a well-formed Persian split light verb

[2] Since Perl 6 is currently under development, the code may need minor modifications in the future. Productions in Perl 6 grammars are written as, for example: `rule S { <NP> <VP> }`

```
S            →  NP-1S VP-1S | NP-2S VP-2S | ...
NP-1S        →  N-1S
NP-2S        →  N-2S
⋮
VP-1S        →  NV-kard V-1S-kard′ | NV-zad V-1S-zad′ | ...
VP-2S        →  NV-kard V-2S-kard′ | NV-zad V-2S-zad′ | ...
⋮
V-1S-kard′   →  AUX-1S V-kard
V-2S-kard′   →  AUX-2S V-kard
⋮
V-1S-zad′    →  AUX-1S V-zad
V-2S-zad′    →  AUX-2S V-zad
⋮
```

Figure 4.2: A non-overgenerating CFG for Persian LVC crossing dependencies

construction. Appendix B shows a similar acceptor for the split passive constructions. Both programs use a transliteration scheme defined in the second column of Appendix C.

The resulting derivation of a context-free grammar that includes feature restrictions is found in the first tree of Figure 4.3. An alternative way to express this is to place the features below the non-terminal's general part-of-speech, as is common in GPSG. The lower tree in Figure 4.3 displays this modified, but equivalent, derivation.

An important assumption that context-free grammars make (and indeed string-rewriting grammars of any expressive power) is that the number of terminals is finite. In grammars for natural languages the terminals are normally words or morphemes. Hopcroft and Ullman (1979, pg. 79) define a CFG as a 4-tuple "$G = (V, T, P, S)$, where V and T are finite sets of *variables* and *terminals*" (italics original). To complete the definition, P is the finite set of productions, or rules, and S is the start symbol, or topmost non-terminal node. However, Gazdar and Pullum (1985, § 2.1.1) note the implausibility of a finite lexicon:

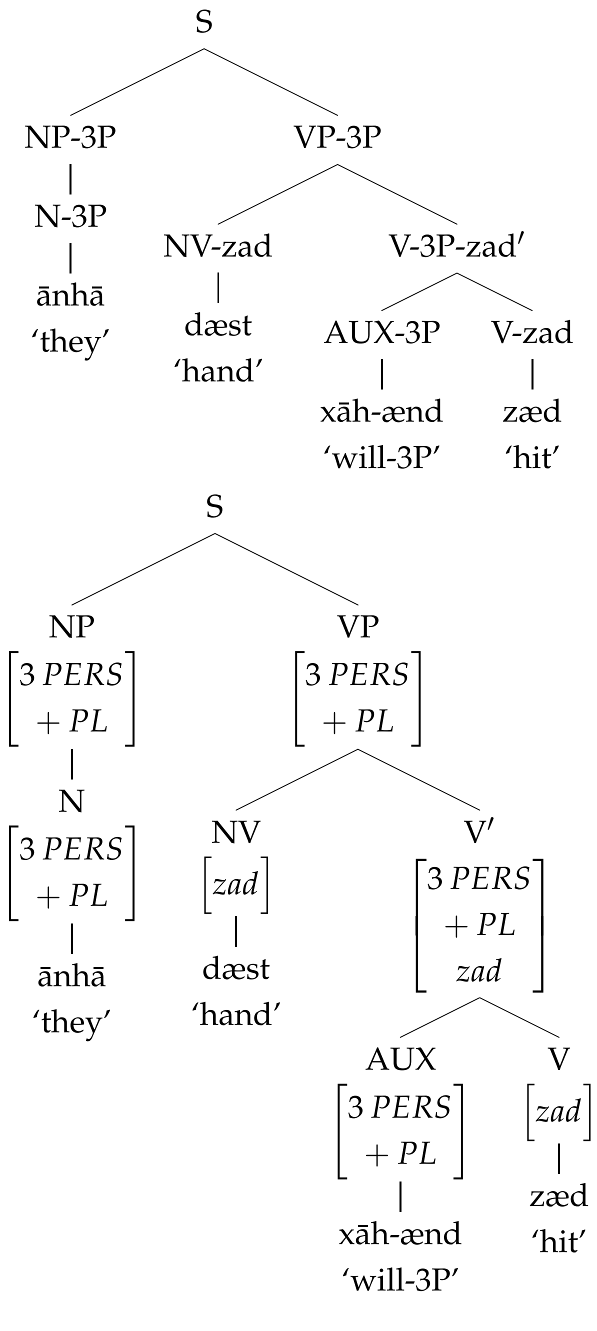


Figure 4.3: Equivalent derived trees from a non-overgenerating CFG grammar

> Do all [natural] languages have a finite lexicon? The common sense answer is “yes”; after all, dictionaries contain all the words in a language, and while dictionaries may be very long [...], they are not infinitely long. But the common sense answer is incorrect: there are few if any languages whose dictionaries contain all the words of the language. No Finnish dictionary contains all the possible forms of Finnish verbs [...] Most languages employ word-formation processes that can apply iteratively to each other’s output, and, in so doing, trivially induce an infinite language [...].

Such is the case with Persian nominal compounds and with light verbs, which occasionally apply word-formation processes from light verb constructions. Thus what were originally two terminal nodes become a single terminal. Iterative nominal compounding is quite common in many languages, and is found in Persian as well:

(3) a. rāh-row
path-go
‘corridor’

b. sær-pušid-e
head-cover-PSPT
‘porch’

c. rāh-row sær-pušid-e
path-go head-cover-PSPT
‘cloister’

Perhaps more interesting is the iterative process for developing new light verbs from light verb constructions. While they are not as productive as the nominal counterpart, they offer challenges to clearly distinguishing terminals from non-terminals:

(4) a. peydā kærdæn
visible do
‘to find’

b. šib peydā-kærdæn
slope visible-do
'to decline'

c. loknæt-e zæbān peydā-kærdæn
stutter-GEN tongue visible-do
'to falter'

(5) a. kār kærdæn
work do
'to work'

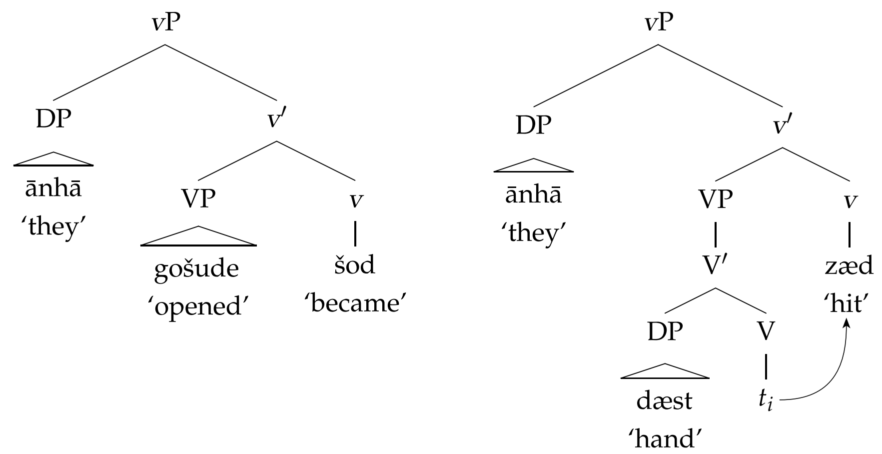
b. sæxt kār-kærdæn
hard work-do
'to grind'

One method of parsing a context-free language that contains non-finite terminals is to write a CFG for individual words/morphemes and another for the entire sentence (Sadock, 1985). The finite number of word-level root nodes would serve as terminals in the sentential CFG. A given sentence would be grammatical if both CFGs accepted their respective inputs as grammatical. The appeal of using a context-free grammar is the formal weakness and computational efficiency. These grammars are fairly high in the Chomsky hierarchy, and the most time a CFG requires to parse a sentence is proportional to the cubed length of the sentence (Earley, 1970). One drawback is that CFGs do not correspond semantically related sentences. Productions would be completely different for crossing dependencies and their related counterparts not in the future tense.

4.2 Minimalist Syntax

The Minimalist Program (Chomsky, 1995) can provide an alternative explanation for crossing dependencies in Persian. The *v*P shell gives a suitable locus for analyzing passive constructions and LVCs, as Megerdoomian (2002) discussed. Hence, the light verbs and the passive verb *šodæn* are found at the *v* node:

(6)



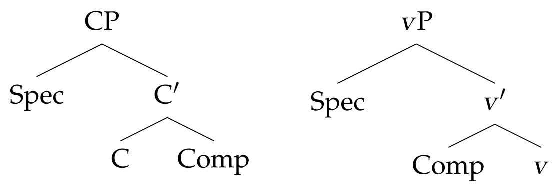
As LVCs themselves may passivize, the light verb must originate as V, then move to *v* if unoccupied. Once reaching this node, the *v* does not overtly move anywhere else.

4.2.1 Split Headedness

At this point I depart from the traditionally-held notion that Persian is head-final. I propose that there is a split-headedness in Persian, where the *v*P node and lower phrases are head-final, and the CP node and lower phrases until *v*P are head-initial. Such an analysis allows us to economically account for the crossing dependencies, as well as many other seemingly-contradictory phenomena in this language.

The general structure of the two phrase types is as follows:

(7)



These two phrase types (CP and *v*P) correspond to the two *phases* mentioned in Chomsky (2000). He describes phases as self-contained components of derivation, and asserts that internal elements of a given phase must be on its phase's edge

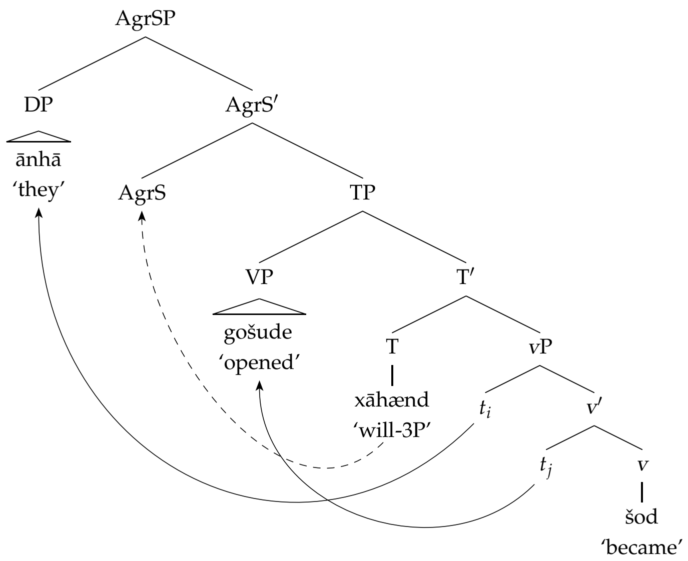


Figure 4.4: A Minimalist tree for Persian split passives

before moving out to another phase. As such, we could say that in this analysis the specifier of *v*P, on the left edge of this phase, moves to the specifier of AgrSP. With this movement, subject agreement features can then be covertly checked at AgrS. This also has the effect of placing the VP on the edge of its phase and allowing it to move outside to the specifier of TP, as can be seen in the split passivization of Figure 4.4 and the split light verb construction of Figure 4.5.

Another benefit of describing CP as head-initial is that this can economically account for four facts about CPs in Persian: 1) complement clauses follow matrix clauses; 2) relative clauses follow matrix clauses; 3) the interrogative particles *āyā/mægær* begin interrogative sentences; and 4) although *wh*-words do not always move, when they do they move to the beginning of a sentence. These phenomena are respectively shown below:

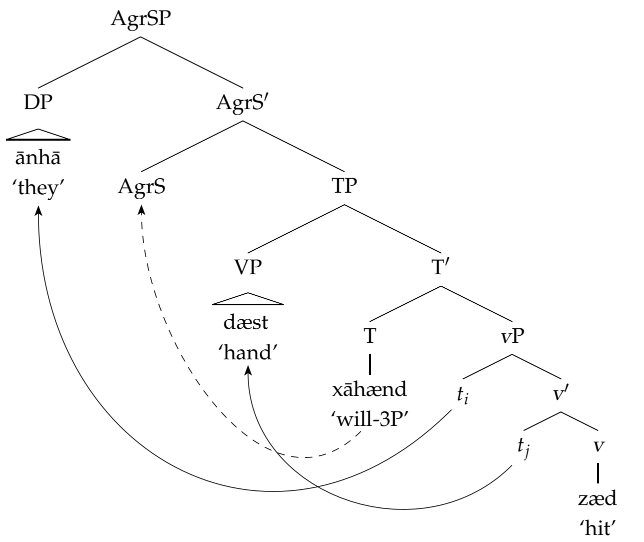


Figure 4.5: A Minimalist tree for Persian split LVCs

(8) a. ne-mi-dun-e [$_{CP}$ ke færdā mi-yām]
NEG-DUR-know-3S that tomorrow DUR-come-1S
'She doesn't know I'm coming tomorrow.' (from Mahootian (1997, pg. 90))

b. un mærd-o [$_{CP}$ ke ruznāme mi-xund] peydā kærd
that man-OM that newspaper DUR-read visible did
'He found the man who was reading the newspaper.' (*Ibid*, pg. 34)

c. āyā in gorbe-ye šomā-st?
INTER this cat-GEN you-is
'Is this your cat?' (*Ibid*, pg. 9)

d. čerā mā sāket be-mān-im?
why we quiet SBJN-remain-1P
'Why do we remain quiet?'

The notion of split headedness was independently noted for Pashto, a closely related language, by Roberts (2000, pgs. 54–63). His division for Pashto headedness loosely approximates to the one for Persian suggested in this paper, although he separates the phrases by functional *vs.* lexical category, rather than structural categories as is suggested here.

By separating the phrases by functional *vs.* lexical, we are able to see a general division between the head-initial functional categories and the head-final lexical categories. On the other hand, by separating the phrases by structure, we can see a general division between the upper head-initial CP-phase and the lower head-final *v*P-phase. It should be noted, however, that both divisions have their exceptions—aspect, for example, proves problematic for both types of divisions (and in both Persian and Pashto).

The Minimalist Program allows us to overcome some of the inherent inadequacies of context-free grammars, such as being able to relate present-tense and future-tense light verb constructions, but at a price. It is not clear at the present time exactly what formal language a grammar for Chomsky (1995)'s Minimalist Program could generate, but it would necessarily be more formally powerful than a context-free language[3].

4.3 Tree Adjoining Grammar

Maclachlan and Rambow (2003) present a TAG[4] analysis of Tagalog CSDs, arguing that adjunction is ideally suited for crossing dependencies. Adjunction allows for an unbounded number of *localized* crossing dependencies, giving a formally restricted account of CSDs.

A general format for adjunction in CSDs may be seen in Figure 4.6, adapted from Maclachlan and Rambow (2003, pg. 102). This would be used for verb-final CSDs, while a mirrored counterpart would be suitable for verb-initial CSDs. A tree may adjoin if it has a leaf node that matches its root node, as in tree (β). It may adjoin to another tree that has a node labeled the same as the adjoining tree's root node. A tree resulting from adjunction may be seen in (γ). The initial and

[3]But see Michaelis (1998) for a characterization of Stabler (1997)'s Minimalist Grammars as mildly context-sensitive.

[4]See Abeillé and Rambow (2000) for an introduction to Tree Adjoining Grammar.

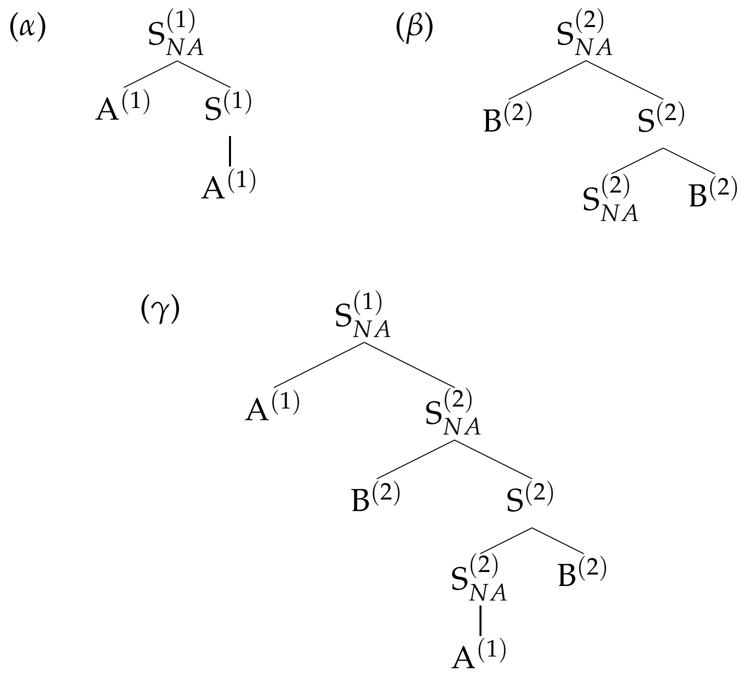


Figure 4.6: TAG initial (α), auxiliary (β), and derived tree (γ) for verb-final CSDs

auxiliary trees in Figure 4.6 generate a language $L = \{xx \mid x \in \{a, b\}^+\}$, which is a reduplicating language.

Figure 4.7 shows a specific format for initial and auxiliary trees in Persian crossing dependencies. Using auxiliary trees in this format allows us to define individual light verb constructions in the lexicon as paired units of a single lexeme. This corresponds with our intuitive notion of pairing them together lexically, while also allowing them to separate in a formally restricted manner.

Adjoining the initial tree with the auxiliary tree gives the derived tree in Figure 4.8. One shortcoming of this approach, however, is that it fails to show proper relations between words. On the other hand, TAG is well understood formally and is known to be mildly context-sensitive. The most time a TAG requires to parse a sentence is proportional to the length of the sentence raised to the sixth (Vijay-Shanker, 1987).

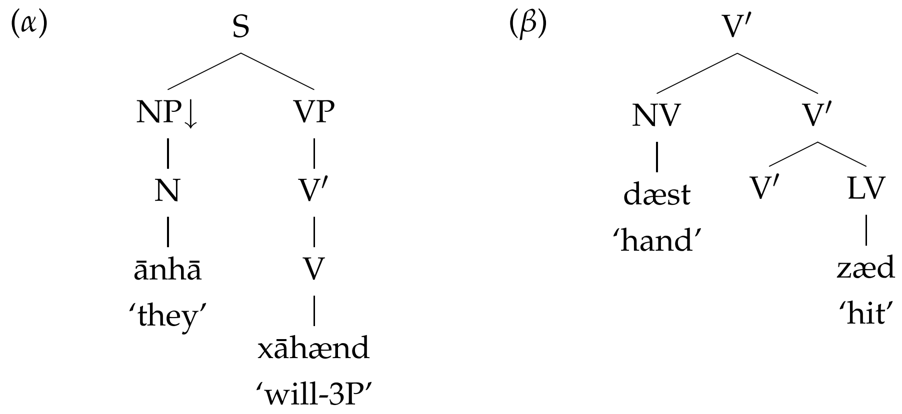


Figure 4.7: A TAG initial tree (α) and auxiliary tree (β)

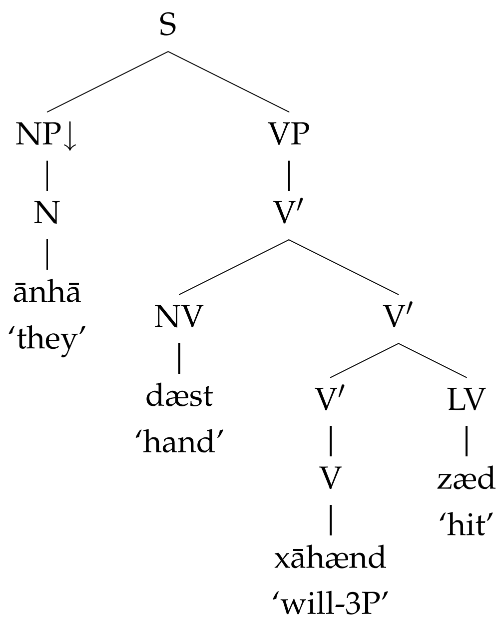


Figure 4.8: A TAG derived tree

4.4 Comparison of Different Crossing Dependencies

The Persian crossing dependencies described in this thesis have properties that are similar to the cross-serial dependencies of Dutch, Swiss German, and Tagalog. Their crossing is not a result of stylistic extraction. Rather, these crossings occur in a highly controlled and predictable manner. We also see similarities with the English example of (2), where both the Persian and English structures have an upper bound on the number of dependencies which can cross. This upper bound distinguishes them from their serial counterparts. The number of dependencies which can cross appears to be limited to one pair in Persian. In contrast to the languages with CSDs, the English and Persian crossing dependencies are required to be in their respective word orders; a non-crossing variant is not an option.

While Swiss German and Dutch CSDs are unbounded in the number of crossing dependencies, Persian LVC crossing dependencies appear unbounded in individual crossing dependency complexity—there is no upper bound on the number of possible light verbs and possible non-verbal elements. In practice, though, both have practical limits. Performance limitations place Swiss German CSDs at a maximum length of about four or five (Shieber, 1985, pg. 341), while the actual number of Persian light verbs in common use is less than 40. Thus five Swiss German cross-serial dependencies results in $2^5 = 32$ possibilities, which can be generalized to

(9) $$\{\text{DAT,ACC}\}^n$$

where n = number of CSDs in a given complement phrase. A single Persian LVC crossing dependency gives $3 \times 2 \times 35 = 210$ possibilities, or

(10) $$[\{1,2,3\}\ \text{Person}] \times [\{SG, PL\}\ \text{Number}] \times [n\ \text{Light Verb}]$$

where n = number of light verbs. With each additional light verb, the possibilities grow $6n$. Thus in theory, the Swiss German CSDs seem more challenging since

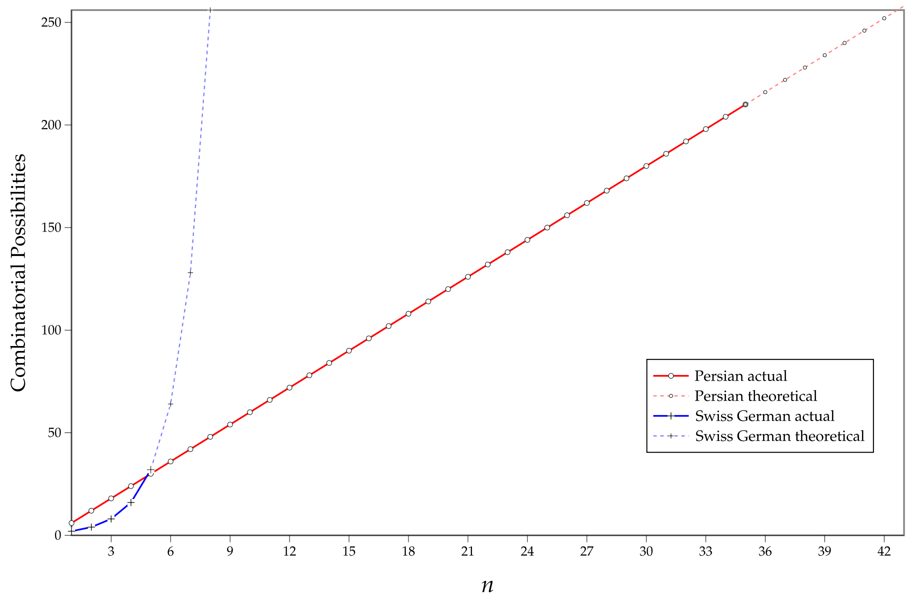


Figure 4.9: Growth of Swiss German and Persian crossing dependencies

they have exponential growth. However, the practical limitations of both language constructions means that the Persian crossing dependencies almost always have more combinatorial possibilities in practice, which is shown in Figure 4.9.

Unlike Persian LVC crossing dependencies, the combination possibilities of Dutch and Swiss German CSDs represent a complete bipartite graph. That is, any noun can *grammatically* combine with any complementizing verb to form a dependency in a CSD, with Swiss German adding the requirement of overt case agreement, either dative or accusative, between a (sometimes optional) noun and its governing verb. This can be represented as a graph which connects every vertex or node from one set or column to every vertex in another set, as seen in the Dutch and Swiss German parts of Figure 4.10. The combination possibilities in Persian light verb constructions are arbitrary, as represented in the Persian part of Figure 4.10. At least with intransitive verbs, not any noun can grammatically combine with any light verb.

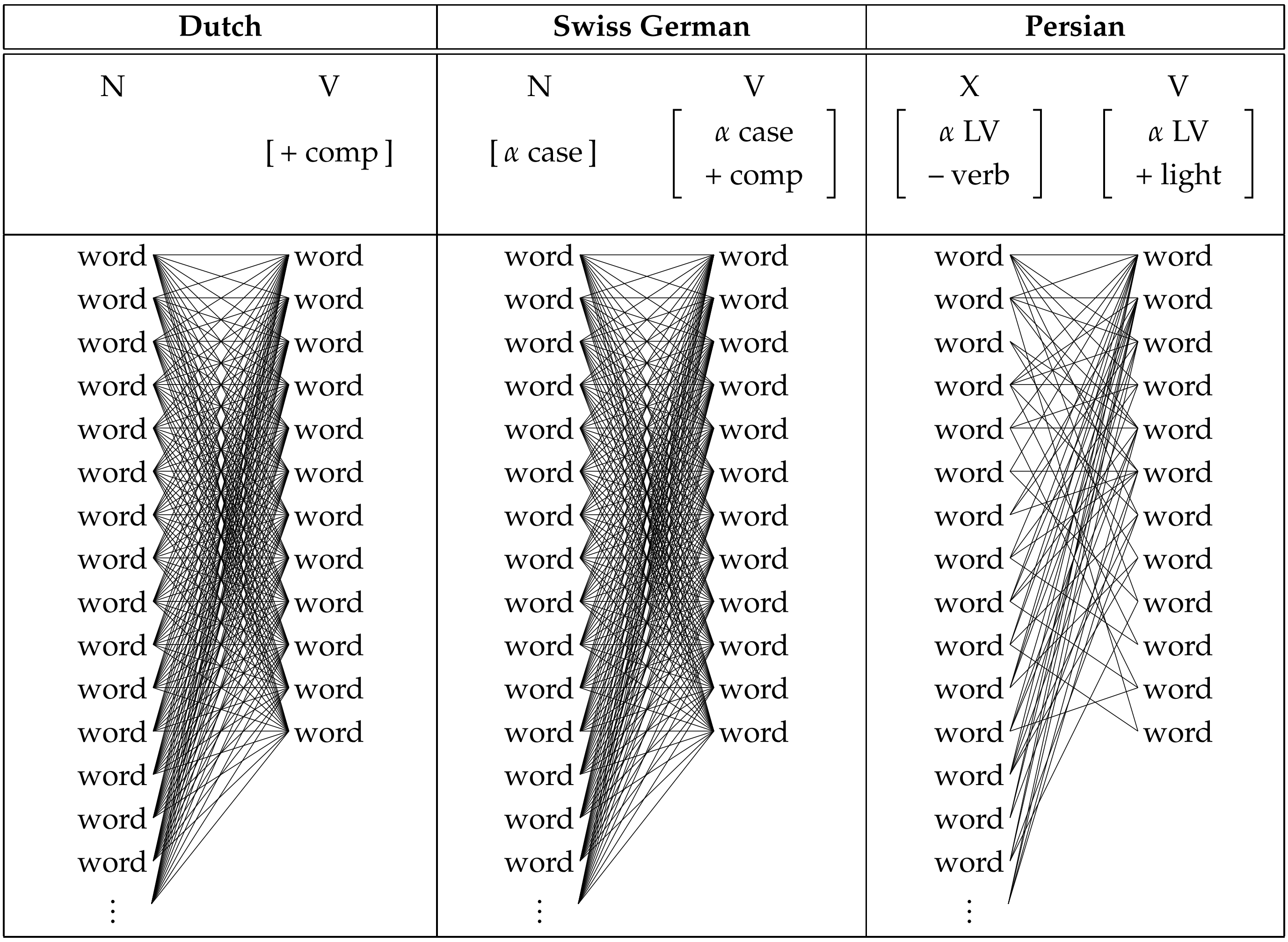


Figure 4.10: Differences among various crossing dependencies

This knowledge would have an impact on the implementation of a system that parses or recognizes these constructions. For example, any noun phrase and any complementizing verb found in their respective positions results in the Dutch construction being grammatical. Less information is required, so the implementation is simpler. Swiss German also allows any noun phrase and any complementizing verb, but requires the two parts to agree in case, as specified lexically with the verb. This means that the implementation of parsing these Swiss German forms is more involved, and each additional crossing dependency doubles the information needed to determine whether the construction is grammatical or not. On the other hand, not any Persian non-verbal element (noun, adjective, etc.) may combine with any light verb. These arbitrary relations require much more information in the grammar, as each pairing must be explicitly specified. This makes the parsing or recognition of these structures particularly complex.

Chapter 5

Conclusion

This thesis has described two varieties of crossing dependencies in the Persian language, namely light verb constructions split by a future auxiliary and passive constructions split by a future auxiliary. These structures are interesting linguistically and computationally because they are not strongly context-free and feature split-headedness. This paper has also shown ways to account for these structures in a context-free grammar, the Minimalist Program, and Tree Adjoining Grammar.

Since both structures are bounded in length, a context-free grammar can recognize both varieties, although the LVC implementation is quite lengthy. A Minimalist explanation would involve overt movement of the specifier of *v*P to the specifier of AgrSP, overt movement of the VP to the specifier of TP, and feature checking of T with AgrS. The same manner which the TAG formalism has been able to locally handle other languages' cross-serial dependencies can nicely account for Persian crossing dependencies.

Persian crossing dependencies resemble Dutch, Swiss German, and Tagalog cross-serial dependencies with two notable differences. First, Persian seems to have an upper bound on the number of crossing dependencies. Second, the amount of crossing information in Persian can be much greater than the CSDs of the other languages. Thus, Persian crossing dependencies are a unique and interesting departure from other types of crossing dependencies that have been previously studied.

Moreover, additional questions arise from the inquiries of this paper. For example, corpus analyses indicate that the modal auxiliary verb *tævānestæn* 'can' may also split light verb constructions, in a different manner than the future auxiliary does. Many other Indo-Iranian languages have light verb constructions, so it would be interesting to see if similar crossing dependencies are found among them. Further investigations could affirm the split-headedness hypothesis proposed in this paper, or could explore more orthodox solutions to the phenomena. Future work in this area could include implementing split-headedness within a Minimalist parser. This thesis will hopefully prompt further research into these areas.

Bibliography

Abeillé, Anne, and Owen Rambow. 2000. *Tree Adjoining Grammars*. Stanford, CA: CSLI.

Aho, Alfred. 1968. Indexed grammars—an extension of context-free grammars. *Journal of the Association for Computing Machinery* 15.647–671.

Bach, Emmon. 1974. *Syntactic Theory*. New York: Holt Rinehard and Winston.

Bar-Hillel, Yehoshua, and Eli Shamir. 1960. Finite state languages: Formal representations and adequacy problems. *Language and Information. Addison Wesley, Reading, Massachusetts* 87–98.

Barendregt, Henk. 1997. The impact of the lambda calculus in logic and computer science. *The Bulletin of Symbolic Logic* 3.181–215.

Bresnan, Joan. 1978. A realistic transformational grammar. In *Linguistic Theory and Psychological Reality*, ed. by M. Halle, J. Bresnan, and G. Miller. Cambridge, MA: MIT Press.

Bresnan, Joan, Ronald Kaplan, Stanley Peters, and Annie Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry* 13.613–35.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 2.113–124.

Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and Control* 2.137–167.

Chomsky, Noam. 1965. *Aspects of the Theory of Syntax*. Cambridge, Massachusetts: The MIT Press.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, Massachusetts: The MIT Press.

Chomsky, Noam. 2000. Minimalist inquiries: The framework. In *Step by Step: Essays on Minimalist Syntax in honor of Howard Lasnik*, ed. by R. Martin, D. Michaels, and J. Uriagereka. Cambridge, MA: MIT Press.

Dehdari, Jon, and Deryle Lonsdale. 2007, forthcoming. A link grammar parser for Persian. In *Aspects of Iranian Linguistics*. Cambridge Scholars Press.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 13.94–102.

Friedl, Jeffrey E. F., and Andy Oram. 2002. *Mastering Regular Expressions*. O'Reilly & Associates, Inc.

Gazdar, Gerald, and Christopher Mellish. 1989. *Natural language processing in Prolog*. Reading, MA: Addison-Wesley.

Gazdar, Gerald, and Geoffrey K. Pullum. 1985. Computationally relevant properties of natural languages and their grammars. *New Generation Computing* 3.273–306.

Harris, Zellig. 1952. Discourse analysis. *Language* 28.1–30.

Hopcroft, John, and Jeffrey Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.

Huybregts, Marinus A. C. 1976. Overlapping dependencies in Dutch. *Utrecht Working Papers in Linguistics* 1.24–65.

Huybregts, Riny. 1984. The weak inadequacy of context-free phrase structure grammars. In *Van Periferie naar Kern*, ed. by Ger de Haan, M. Trommelen, and W. Zonneveld, 81–99. Dordrecht: Foris.

Joshi, Aravind, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computing and System Sciences* 10.136–163.

Karimi, Simin (ed.) 2003. *Word Order and Scrambling*. Oxford, UK: Blackwell Publishing.

Kracht, Marcus. 2003. *The Mathematics of Language*. Berlin: Mouton de Gruyter.

Maclachlan, Anna, and Owen Rambow. 2003. Cross-serial dependencies in Tagalog. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, 100–104, Università di Venezia.

Mahootian, Shahrzad. 1997. *Persian*. Descriptive Grammars. London: Routledge.

Manaster Ramer, Alexis. 1988. Review of "The formal complexity of natural language" by Savitch et al. (1987). *Computational Linguistics* 14.98–103.

Markov, Andrey A. 1960. The theory of algorithms. *American Mathematical Society Translations* 2.1–14.

Megerdoomian, Karine, 2002. *Beyond Words and Phrases: A Unified Theory of Predicate Composition*. University of Southern California dissertation.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. *Logical Aspects of Computational Linguistics, Lecture Notes in Artificial Intelligence* 2014.179–182.

Partee, Barbara, Alice ter Meulen, and Robert Wall. 1990. *Mathematical Methods in Linguistics*. Dordrecht: Kluwer Academic Publishers.

Pollard, Carl, 1984. *Generalized phrase structure grammars, head grammars, and natural language*. Stanford University dissertation.

Postal, Paul. 1964. Limitations of phrase structure grammars. In *The Structure of Language: Readings in the Philosophy of Language*, ed. by J. Fodor and J. Katz, 137–151. Englewood Cliffs, N.J.: Prentice-Hall.

Pullum, Geoffrey K., and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy* 4.471–504.

Roberts, Taylor, 2000. *Clitics and Agreement*. The Massachusetts Institute of Technology dissertation.

Sadock, Jerrold M. 1985. Autolexical syntax: A theory of noun incorporation and similar phenomena. *Natural Language and Linguistic Theory* 3.379–439.

Savitch, Walter. 1987. Context-sensitive grammar and natural language syntax. In *The Formal Complexity of Natural Language*, ed. by W. J. Savitch, E. Bach, W. Marsh, and G. Safran-Naveh, 358–368. Dordrecht, Holland: Reidel.

Shieber, Stuart. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8.333–343.

Sipser, Michael. 1997. *Introduction to the Theory of Computation*. Boston, MA: PWS Publishing.

Stabler, Edward. 1997. Derivational minimalism. *Logical Aspects of Computational Linguistics* 1328.68–95.

Stabler, Edward. 2004. Varieties of crossing dependencies. *Cognitive Science* 28.699–720.

Steedman, Mark. 1985. Dependency and coordination in the grammar of Dutch and English. *Language* 61.523–568.

Turing, Alan. 1937. Computability and lambda definability. *Journal of Symbolic Logic* 42.230–265.

Vijay-Shanker, K., 1987. *A Study of Tree Adjoining Grammars*. University of Pennsylvania dissertation.

Vogel, Carl, Ulrike Hahn, and Holly Branigan. 1996. Cross-serial dependencies are not hard to process. In *Proceedings of COLING-96*, 157–162, Copenhagen, Denmark.

Appendix A

A Perl 6 Grammar for Persian LVC Crossing Dependencies

```
#!/usr/bin/pugs

### Jon Dehdari, 2006
### A Perl 6 context-free grammar to recognize Persian LVC crossing dependencies

use v6;

$_ = shift || "AnhA dst xuAhnd zd";

grammar Persian {
  # Grammatical stuff
  rule sentence  {
      <NP_1S> <VP_1S> | <NP_2S> <VP_2S> | <NP_3S> <VP_3S> | <NP_1P> <VP_1P> | <NP_2P> <VP_2P> | <NP_3P> <VP_3P>
  }

  rule VP_1S:w   {
      <NV_krd> <Vbar_1S_krd> | <NV_Cd> <Vbar_1S_Cd> | <NV_dAd> <Vbar_1S_dAd> | <NV_zd> <Vbar_1S_zd> |
      <NV_bud> <Vbar_1S_bud> | <NV_dACt> <Vbar_1S_dACt> | <NV_grft> <Vbar_1S_grft> |
      <NV_rft> <Vbar_1S_rft> | <NV_kCid> <Vbar_1S_kCid> | <NV_andAxt> <Vbar_1S_andAxt> |
      <NV_xurd> <Vbar_1S_xurd> | <NV_gLACt> <Vbar_1S_gLACt> | <NV_Aurd> <Vbar_1S_Aurd> |
      <NV_sAxt> <Vbar_1S_sAxt> | <NV_gft> <Vbar_1S_gft> | <NV_iAft> <Vbar_1S_iAft> |
      <NV_bst> <Vbar_1S_bst> | <NV_brd> <Vbar_1S_brd> | <NV_Amd> <Vbar_1S_Amd> |
      <NV_rixt> <Vbar_1S_rixt> | <NV_oftAd> <Vbar_1S_oftAd> | <NV_nmud> <Vbar_1S_nmud> |
      <NV_rsAnd> <Vbar_1S_rsAnd> | <NV_brdACt> <Vbar_1S_brdACt> | <NV_jst> <Vbar_1S_jst> |
      <NV_bxCid> <Vbar_1S_bxCid> | <NV_xuAnd> <Vbar_1S_xuAnd> | <NV_pidAkrd> <Vbar_1S_pidAkrd> |
      <NV_rsid> <Vbar_1S_rsid> | <NV_did> <Vbar_1S_did> | <NV_bAxt> <Vbar_1S_bAxt> |
      <NV_kubid> <Vbar_1S_kubid> | <NV_grdid> <Vbar_1S_grdid> | <NV_cid> <Vbar_1S_cid> |
      <NV_brid> <Vbar_1S_brid> | ...
  }

  rule VP_2S:w   {
      <NV_krd> <Vbar_2S_krd> | <NV_Cd> <Vbar_2S_Cd> | <NV_dAd> <Vbar_2S_dAd> | <NV_zd> <Vbar_2S_zd> |
      <NV_bud> <Vbar_2S_bud> | <NV_dACt> <Vbar_2S_dACt> | <NV_grft> <Vbar_2S_grft> |
      <NV_rft> <Vbar_2S_rft> | <NV_kCid> <Vbar_2S_kCid> | <NV_andAxt> <Vbar_2S_andAxt> |
      <NV_xurd> <Vbar_2S_xurd> | <NV_gLACt> <Vbar_2S_gLACt> | <NV_Aurd> <Vbar_2S_Aurd> |
      <NV_sAxt> <Vbar_2S_sAxt> | <NV_gft> <Vbar_2S_gft> | <NV_iAft> <Vbar_2S_iAft> |
      <NV_bst> <Vbar_2S_bst> | <NV_brd> <Vbar_2S_brd> | <NV_Amd> <Vbar_2S_Amd> |
      <NV_rixt> <Vbar_2S_rixt> | <NV_oftAd> <Vbar_2S_oftAd> | <NV_nmud> <Vbar_2S_nmud> |
      <NV_rsAnd> <Vbar_2S_rsAnd> | <NV_brdACt> <Vbar_2S_brdACt> | <NV_jst> <Vbar_2S_jst> |
```

```
    <NV_bxCid> <Vbar_2S_bxCid> | <NV_xuAnd> <Vbar_2S_xuAnd> | <NV_pidAkrd> <Vbar_2S_pidAkrd> |
    <NV_rsid> <Vbar_2S_rsid> | <NV_did> <Vbar_2S_did> | <NV_bAxt> <Vbar_2S_bAxt> |
    <NV_kubid> <Vbar_2S_kubid> | <NV_grdid> <Vbar_2S_grdid> | <NV_cid> <Vbar_2S_cid> |
    <NV_brid> <Vbar_2S_brid> | ...
}

rule VP_3S:w   {
    <NV_krd> <Vbar_3S_krd> | <NV_Cd> <Vbar_3S_Cd> | <NV_dAd> <Vbar_3S_dAd> | <NV_zd> <Vbar_3S_zd> |
    <NV_bud> <Vbar_3S_bud> | <NV_dACt> <Vbar_3S_dACt> | <NV_grft> <Vbar_3S_grft> |
    <NV_rft> <Vbar_3S_rft> | <NV_kCid> <Vbar_3S_kCid> | <NV_andAxt> <Vbar_3S_andAxt> |
    <NV_xurd> <Vbar_3S_xurd> | <NV_gLACt> <Vbar_3S_gLACt> | <NV_Aurd> <Vbar_3S_Aurd> |
    <NV_sAxt> <Vbar_3S_sAxt> | <NV_gft> <Vbar_3S_gft> | <NV_iAft> <Vbar_3S_iAft> |
    <NV_bst> <Vbar_3S_bst> | <NV_brd> <Vbar_3S_brd> | <NV_Amd> <Vbar_3S_Amd> |
    <NV_rixt> <Vbar_3S_rixt> | <NV_oftAd> <Vbar_3S_oftAd> | <NV_nmud> <Vbar_3S_nmud> |
    <NV_rsAnd> <Vbar_3S_rsAnd> | <NV_brdACt> <Vbar_3S_brdACt> | <NV_jst> <Vbar_3S_jst> |
    <NV_bxCid> <Vbar_3S_bxCid> | <NV_xuAnd> <Vbar_3S_xuAnd> | <NV_pidAkrd> <Vbar_3S_pidAkrd> |
    <NV_rsid> <Vbar_3S_rsid> | <NV_did> <Vbar_3S_did> | <NV_bAxt> <Vbar_3S_bAxt> |
    <NV_kubid> <Vbar_3S_kubid> | <NV_grdid> <Vbar_3S_grdid> | <NV_cid> <Vbar_3S_cid> |
    <NV_brid> <Vbar_3S_brid> | ...
}

rule VP_1P:w   {
    <NV_krd> <Vbar_1P_krd> | <NV_Cd> <Vbar_1P_Cd> | <NV_dAd> <Vbar_1P_dAd> | <NV_zd> <Vbar_1P_zd> |
    <NV_bud> <Vbar_1P_bud> | <NV_dACt> <Vbar_1P_dACt> | <NV_grft> <Vbar_1P_grft> |
    <NV_rft> <Vbar_1P_rft> | <NV_kCid> <Vbar_1P_kCid> | <NV_andAxt> <Vbar_1P_andAxt> |
    <NV_xurd> <Vbar_1P_xurd> | <NV_gLACt> <Vbar_1P_gLACt> | <NV_Aurd> <Vbar_1P_Aurd> |
    <NV_sAxt> <Vbar_1P_sAxt> | <NV_gft> <Vbar_1P_gft> | <NV_iAft> <Vbar_1P_iAft> |
    <NV_bst> <Vbar_1P_bst> | <NV_brd> <Vbar_1P_brd> | <NV_Amd> <Vbar_1P_Amd> |
    <NV_rixt> <Vbar_1P_rixt> | <NV_oftAd> <Vbar_1P_oftAd> | <NV_nmud> <Vbar_1P_nmud> |
    <NV_rsAnd> <Vbar_1P_rsAnd> | <NV_brdACt> <Vbar_1P_brdACt> | <NV_jst> <Vbar_1P_jst> |
    <NV_bxCid> <Vbar_1P_bxCid> | <NV_xuAnd> <Vbar_1P_xuAnd> | <NV_pidAkrd> <Vbar_1P_pidAkrd> |
    <NV_rsid> <Vbar_1P_rsid> | <NV_did> <Vbar_1P_did> | <NV_bAxt> <Vbar_1P_bAxt> |
    <NV_kubid> <Vbar_1P_kubid> | <NV_grdid> <Vbar_1P_grdid> | <NV_cid> <Vbar_1P_cid> |
    <NV_brid> <Vbar_1P_brid> | ...
}

rule VP_2P:w   {
    <NV_krd> <Vbar_2P_krd> | <NV_Cd> <Vbar_2P_Cd> | <NV_dAd> <Vbar_2P_dAd> | <NV_zd> <Vbar_2P_zd> |
    <NV_bud> <Vbar_2P_bud> | <NV_dACt> <Vbar_2P_dACt> | <NV_grft> <Vbar_2P_grft> |
    <NV_rft> <Vbar_2P_rft> | <NV_kCid> <Vbar_2P_kCid> | <NV_andAxt> <Vbar_2P_andAxt> |
    <NV_xurd> <Vbar_2P_xurd> | <NV_gLACt> <Vbar_2P_gLACt> | <NV_Aurd> <Vbar_2P_Aurd> |
    <NV_sAxt> <Vbar_2P_sAxt> | <NV_gft> <Vbar_2P_gft> | <NV_iAft> <Vbar_2P_iAft> |
    <NV_bst> <Vbar_2P_bst> | <NV_brd> <Vbar_2P_brd> | <NV_Amd> <Vbar_2P_Amd> |
    <NV_rixt> <Vbar_2P_rixt> | <NV_oftAd> <Vbar_2P_oftAd> | <NV_nmud> <Vbar_2P_nmud> |
    <NV_rsAnd> <Vbar_2P_rsAnd> | <NV_brdACt> <Vbar_2P_brdACt> | <NV_jst> <Vbar_2P_jst> |
    <NV_bxCid> <Vbar_2P_bxCid> | <NV_xuAnd> <Vbar_2P_xuAnd> | <NV_pidAkrd> <Vbar_2P_pidAkrd> |
    <NV_rsid> <Vbar_2P_rsid> | <NV_did> <Vbar_2P_did> | <NV_bAxt> <Vbar_2P_bAxt> |
    <NV_kubid> <Vbar_2P_kubid> | <NV_grdid> <Vbar_2P_grdid> | <NV_cid> <Vbar_2P_cid> |
    <NV_brid> <Vbar_2P_brid> | ...
}

rule VP_3P:w   {
    <NV_krd> <Vbar_3P_krd> | <NV_Cd> <Vbar_3P_Cd> | <NV_dAd> <Vbar_3P_dAd> | <NV_zd> <Vbar_3P_zd> |
```

```
    <NV_bud> <Vbar_3P_bud> | <NV_dACt> <Vbar_3P_dACt> | <NV_grft> <Vbar_3P_grft> |
    <NV_rft> <Vbar_3P_rft> | <NV_kCid> <Vbar_3P_kCid> | <NV_andAxt> <Vbar_3P_andAxt> |
    <NV_xurd> <Vbar_3P_xurd> | <NV_gLACt> <Vbar_3P_gLACt> | <NV_Aurd> <Vbar_3P_Aurd> |
    <NV_sAxt> <Vbar_3P_sAxt> | <NV_gft> <Vbar_3P_gft> | <NV_iAft> <Vbar_3P_iAft> |
    <NV_bst> <Vbar_3P_bst> | <NV_brd> <Vbar_3P_brd> | <NV_Amd> <Vbar_3P_Amd> |
    <NV_rixt> <Vbar_3P_rixt> | <NV_oftAd> <Vbar_3P_oftAd> | <NV_nmud> <Vbar_3P_nmud> |
    <NV_rsAnd> <Vbar_3P_rsAnd> | <NV_brdACt> <Vbar_3P_brdACt> | <NV_jst> <Vbar_3P_jst> |
    <NV_bxCid> <Vbar_3P_bxCid> | <NV_xuAnd> <Vbar_3P_xuAnd> | <NV_pidAkrd> <Vbar_3P_pidAkrd> |
    <NV_rsid> <Vbar_3P_rsid> | <NV_did> <Vbar_3P_did> | <NV_bAxt> <Vbar_3P_bAxt> |
    <NV_kubid> <Vbar_3P_kubid> | <NV_grdid> <Vbar_3P_grdid> | <NV_cid> <Vbar_3P_cid> |
    <NV_brid> <Vbar_3P_brid> | ...
}

rule Vbar_1S_krd:w     { <AUX_1S> <LV_krd> }
rule Vbar_2S_krd:w     { <AUX_2S> <LV_krd> }
rule Vbar_3S_krd:w     { <AUX_3S> <LV_krd> }
rule Vbar_1P_krd:w     { <AUX_1P> <LV_krd> }
rule Vbar_2P_krd:w     { <AUX_2P> <LV_krd> }
rule Vbar_3P_krd:w     { <AUX_3P> <LV_krd> }
rule Vbar_1S_Cd:w      { <AUX_1S> <LV_Cd> }
rule Vbar_2S_Cd:w      { <AUX_2S> <LV_Cd> }
rule Vbar_3S_Cd:w      { <AUX_3S> <LV_Cd> }
rule Vbar_1P_Cd:w      { <AUX_1P> <LV_Cd> }
rule Vbar_2P_Cd:w      { <AUX_2P> <LV_Cd> }
rule Vbar_3P_Cd:w      { <AUX_3P> <LV_Cd> }
rule Vbar_1S_dAd:w     { <AUX_1S> <LV_dAd> }
rule Vbar_2S_dAd:w     { <AUX_2S> <LV_dAd> }
rule Vbar_3S_dAd:w     { <AUX_3S> <LV_dAd> }
rule Vbar_1P_dAd:w     { <AUX_1P> <LV_dAd> }
rule Vbar_2P_dAd:w     { <AUX_2P> <LV_dAd> }
rule Vbar_3P_dAd:w     { <AUX_3P> <LV_dAd> }
rule Vbar_1S_zd:w      { <AUX_1S> <LV_zd> }
rule Vbar_2S_zd:w      { <AUX_2S> <LV_zd> }
rule Vbar_3S_zd:w      { <AUX_3S> <LV_zd> }
rule Vbar_1P_zd:w      { <AUX_1P> <LV_zd> }
rule Vbar_2P_zd:w      { <AUX_2P> <LV_zd> }
rule Vbar_3P_zd:w      { <AUX_3P> <LV_zd> }
rule Vbar_1S_bud:w     { <AUX_1S> <LV_bud> }
rule Vbar_2S_bud:w     { <AUX_2S> <LV_bud> }
rule Vbar_3S_bud:w     { <AUX_3S> <LV_bud> }
rule Vbar_1P_bud:w     { <AUX_1P> <LV_bud> }
rule Vbar_2P_bud:w     { <AUX_2P> <LV_bud> }
rule Vbar_3P_bud:w     { <AUX_3P> <LV_bud> }
rule Vbar_1S_dACt:w    { <AUX_1S> <LV_dACt> }
rule Vbar_2S_dACt:w    { <AUX_2S> <LV_dACt> }
rule Vbar_3S_dACt:w    { <AUX_3S> <LV_dACt> }
rule Vbar_1P_dACt:w    { <AUX_1P> <LV_dACt> }
rule Vbar_2P_dACt:w    { <AUX_2P> <LV_dACt> }
rule Vbar_3P_dACt:w    { <AUX_3P> <LV_dACt> }
rule Vbar_1S_grft:w    { <AUX_1S> <LV_grft> }
rule Vbar_2S_grft:w    { <AUX_2S> <LV_grft> }
rule Vbar_3S_grft:w    { <AUX_3S> <LV_grft> }
rule Vbar_1P_grft:w    { <AUX_1P> <LV_grft> }
```

```
rule Vbar_2P_grft:w    { <AUX_2P> <LV_grft> }
rule Vbar_3P_grft:w    { <AUX_3P> <LV_grft> }
rule Vbar_1S_rft:w     { <AUX_1S> <LV_rft> }
rule Vbar_2S_rft:w     { <AUX_2S> <LV_rft> }
rule Vbar_3S_rft:w     { <AUX_3S> <LV_rft> }
rule Vbar_1P_rft:w     { <AUX_1P> <LV_rft> }
rule Vbar_2P_rft:w     { <AUX_2P> <LV_rft> }
rule Vbar_3P_rft:w     { <AUX_3P> <LV_rft> }
rule Vbar_1S_kCid:w    { <AUX_1S> <LV_kCid> }
rule Vbar_2S_kCid:w    { <AUX_2S> <LV_kCid> }
rule Vbar_3S_kCid:w    { <AUX_3S> <LV_kCid> }
rule Vbar_1P_kCid:w    { <AUX_1P> <LV_kCid> }
rule Vbar_2P_kCid:w    { <AUX_2P> <LV_kCid> }
rule Vbar_3P_kCid:w    { <AUX_3P> <LV_kCid> }
rule Vbar_1S_andAxt:w  { <AUX_1S> <LV_andAxt> }
rule Vbar_2S_andAxt:w  { <AUX_2S> <LV_andAxt> }
rule Vbar_3S_andAxt:w  { <AUX_3S> <LV_andAxt> }
rule Vbar_1P_andAxt:w  { <AUX_1P> <LV_andAxt> }
rule Vbar_2P_andAxt:w  { <AUX_2P> <LV_andAxt> }
rule Vbar_3P_andAxt:w  { <AUX_3P> <LV_andAxt> }
rule Vbar_1S_xurd:w    { <AUX_1S> <LV_xurd> }
rule Vbar_2S_xurd:w    { <AUX_2S> <LV_xurd> }
rule Vbar_3S_xurd:w    { <AUX_3S> <LV_xurd> }
rule Vbar_1P_xurd:w    { <AUX_1P> <LV_xurd> }
rule Vbar_2P_xurd:w    { <AUX_2P> <LV_xurd> }
rule Vbar_3P_xurd:w    { <AUX_3P> <LV_xurd> }
rule Vbar_1S_gLACt:w   { <AUX_1S> <LV_gLACt> }
rule Vbar_2S_gLACt:w   { <AUX_2S> <LV_gLACt> }
rule Vbar_3S_gLACt:w   { <AUX_3S> <LV_gLACt> }
rule Vbar_1P_gLACt:w   { <AUX_1P> <LV_gLACt> }
rule Vbar_2P_gLACt:w   { <AUX_2P> <LV_gLACt> }
rule Vbar_3P_gLACt:w   { <AUX_3P> <LV_gLACt> }
rule Vbar_1S_Aurd:w    { <AUX_1S> <LV_Aurd> }
rule Vbar_2S_Aurd:w    { <AUX_2S> <LV_Aurd> }
rule Vbar_3S_Aurd:w    { <AUX_3S> <LV_Aurd> }
rule Vbar_1P_Aurd:w    { <AUX_1P> <LV_Aurd> }
rule Vbar_2P_Aurd:w    { <AUX_2P> <LV_Aurd> }
rule Vbar_3P_Aurd:w    { <AUX_3P> <LV_Aurd> }
rule Vbar_1S_sAxt:w    { <AUX_1S> <LV_sAxt> }
rule Vbar_2S_sAxt:w    { <AUX_2S> <LV_sAxt> }
rule Vbar_3S_sAxt:w    { <AUX_3S> <LV_sAxt> }
rule Vbar_1P_sAxt:w    { <AUX_1P> <LV_sAxt> }
rule Vbar_2P_sAxt:w    { <AUX_2P> <LV_sAxt> }
rule Vbar_3P_sAxt:w    { <AUX_3P> <LV_sAxt> }
rule Vbar_1S_gft:w     { <AUX_1S> <LV_gft> }
rule Vbar_2S_gft:w     { <AUX_2S> <LV_gft> }
rule Vbar_3S_gft:w     { <AUX_3S> <LV_gft> }
rule Vbar_1P_gft:w     { <AUX_1P> <LV_gft> }
rule Vbar_2P_gft:w     { <AUX_2P> <LV_gft> }
rule Vbar_3P_gft:w     { <AUX_3P> <LV_gft> }
rule Vbar_1S_iAft:w    { <AUX_1S> <LV_iAft> }
rule Vbar_2S_iAft:w    { <AUX_2S> <LV_iAft> }
rule Vbar_3S_iAft:w    { <AUX_3S> <LV_iAft> }
```

```
rule Vbar_1P_iAft:w    { <AUX_1P> <LV_iAft> }
rule Vbar_2P_iAft:w    { <AUX_2P> <LV_iAft> }
rule Vbar_3P_iAft:w    { <AUX_3P> <LV_iAft> }
rule Vbar_1S_bst:w     { <AUX_1S> <LV_bst> }
rule Vbar_2S_bst:w     { <AUX_2S> <LV_bst> }
rule Vbar_3S_bst:w     { <AUX_3S> <LV_bst> }
rule Vbar_1P_bst:w     { <AUX_1P> <LV_bst> }
rule Vbar_2P_bst:w     { <AUX_2P> <LV_bst> }
rule Vbar_3P_bst:w     { <AUX_3P> <LV_bst> }
rule Vbar_1S_brd:w     { <AUX_1S> <LV_brd> }
rule Vbar_2S_brd:w     { <AUX_2S> <LV_brd> }
rule Vbar_3S_brd:w     { <AUX_3S> <LV_brd> }
rule Vbar_1P_brd:w     { <AUX_1P> <LV_brd> }
rule Vbar_2P_brd:w     { <AUX_2P> <LV_brd> }
rule Vbar_3P_brd:w     { <AUX_3P> <LV_brd> }
rule Vbar_1S_Amd:w     { <AUX_1S> <LV_Amd> }
rule Vbar_2S_Amd:w     { <AUX_2S> <LV_Amd> }
rule Vbar_3S_Amd:w     { <AUX_3S> <LV_Amd> }
rule Vbar_1P_Amd:w     { <AUX_1P> <LV_Amd> }
rule Vbar_2P_Amd:w     { <AUX_2P> <LV_Amd> }
rule Vbar_3P_Amd:w     { <AUX_3P> <LV_Amd> }
rule Vbar_1S_rixt:w    { <AUX_1S> <LV_rixt> }
rule Vbar_2S_rixt:w    { <AUX_2S> <LV_rixt> }
rule Vbar_3S_rixt:w    { <AUX_3S> <LV_rixt> }
rule Vbar_1P_rixt:w    { <AUX_1P> <LV_rixt> }
rule Vbar_2P_rixt:w    { <AUX_2P> <LV_rixt> }
rule Vbar_3P_rixt:w    { <AUX_3P> <LV_rixt> }
rule Vbar_1S_oftAd:w   { <AUX_1S> <LV_oftAd> }
rule Vbar_2S_oftAd:w   { <AUX_2S> <LV_oftAd> }
rule Vbar_3S_oftAd:w   { <AUX_3S> <LV_oftAd> }
rule Vbar_1P_oftAd:w   { <AUX_1P> <LV_oftAd> }
rule Vbar_2P_oftAd:w   { <AUX_2P> <LV_oftAd> }
rule Vbar_3P_oftAd:w   { <AUX_3P> <LV_oftAd> }
rule Vbar_1S_nmud:w    { <AUX_1S> <LV_nmud> }
rule Vbar_2S_nmud:w    { <AUX_2S> <LV_nmud> }
rule Vbar_3S_nmud:w    { <AUX_3S> <LV_nmud> }
rule Vbar_1P_nmud:w    { <AUX_1P> <LV_nmud> }
rule Vbar_2P_nmud:w    { <AUX_2P> <LV_nmud> }
rule Vbar_3P_nmud:w    { <AUX_3P> <LV_nmud> }
rule Vbar_1S_rsAnd:w   { <AUX_1S> <LV_rsAnd> }
rule Vbar_2S_rsAnd:w   { <AUX_2S> <LV_rsAnd> }
rule Vbar_3S_rsAnd:w   { <AUX_3S> <LV_rsAnd> }
rule Vbar_1P_rsAnd:w   { <AUX_1P> <LV_rsAnd> }
rule Vbar_2P_rsAnd:w   { <AUX_2P> <LV_rsAnd> }
rule Vbar_3P_rsAnd:w   { <AUX_3P> <LV_rsAnd> }
rule Vbar_1S_brdACt:w  { <AUX_1S> <LV_brdACt> }
rule Vbar_2S_brdACt:w  { <AUX_2S> <LV_brdACt> }
rule Vbar_3S_brdACt:w  { <AUX_3S> <LV_brdACt> }
rule Vbar_1P_brdACt:w  { <AUX_1P> <LV_brdACt> }
rule Vbar_2P_brdACt:w  { <AUX_2P> <LV_brdACt> }
rule Vbar_3P_brdACt:w  { <AUX_3P> <LV_brdACt> }
rule Vbar_1S_jst:w     { <AUX_1S> <LV_jst> }
rule Vbar_2S_jst:w     { <AUX_2S> <LV_jst> }
```

```
rule Vbar_3S_jst:w     { <AUX_3S> <LV_jst> }
rule Vbar_1P_jst:w     { <AUX_1P> <LV_jst> }
rule Vbar_2P_jst:w     { <AUX_2P> <LV_jst> }
rule Vbar_3P_jst:w     { <AUX_3P> <LV_jst> }
rule Vbar_1S_bxCid:w   { <AUX_1S> <LV_bxCid> }
rule Vbar_2S_bxCid:w   { <AUX_2S> <LV_bxCid> }
rule Vbar_3S_bxCid:w   { <AUX_3S> <LV_bxCid> }
rule Vbar_1P_bxCid:w   { <AUX_1P> <LV_bxCid> }
rule Vbar_2P_bxCid:w   { <AUX_2P> <LV_bxCid> }
rule Vbar_3P_bxCid:w   { <AUX_3P> <LV_bxCid> }
rule Vbar_1S_xuAnd:w   { <AUX_1S> <LV_xuAnd> }
rule Vbar_2S_xuAnd:w   { <AUX_2S> <LV_xuAnd> }
rule Vbar_3S_xuAnd:w   { <AUX_3S> <LV_xuAnd> }
rule Vbar_1P_xuAnd:w   { <AUX_1P> <LV_xuAnd> }
rule Vbar_2P_xuAnd:w   { <AUX_2P> <LV_xuAnd> }
rule Vbar_3P_xuAnd:w   { <AUX_3P> <LV_xuAnd> }
rule Vbar_1S_pidAkrd:w { <AUX_1S> <LV_pidAkrd> }
rule Vbar_2S_pidAkrd:w { <AUX_2S> <LV_pidAkrd> }
rule Vbar_3S_pidAkrd:w { <AUX_3S> <LV_pidAkrd> }
rule Vbar_1P_pidAkrd:w { <AUX_1P> <LV_pidAkrd> }
rule Vbar_2P_pidAkrd:w { <AUX_2P> <LV_pidAkrd> }
rule Vbar_3P_pidAkrd:w { <AUX_3P> <LV_pidAkrd> }
rule Vbar_1S_rsid:w    { <AUX_1S> <LV_rsid> }
rule Vbar_2S_rsid:w    { <AUX_2S> <LV_rsid> }
rule Vbar_3S_rsid:w    { <AUX_3S> <LV_rsid> }
rule Vbar_1P_rsid:w    { <AUX_1P> <LV_rsid> }
rule Vbar_2P_rsid:w    { <AUX_2P> <LV_rsid> }
rule Vbar_3P_rsid:w    { <AUX_3P> <LV_rsid> }
rule Vbar_1S_did:w     { <AUX_1S> <LV_did> }
rule Vbar_2S_did:w     { <AUX_2S> <LV_did> }
rule Vbar_3S_did:w     { <AUX_3S> <LV_did> }
rule Vbar_1P_did:w     { <AUX_1P> <LV_did> }
rule Vbar_2P_did:w     { <AUX_2P> <LV_did> }
rule Vbar_3P_did:w     { <AUX_3P> <LV_did> }
rule Vbar_1S_bAxt:w    { <AUX_1S> <LV_bAxt> }
rule Vbar_2S_bAxt:w    { <AUX_2S> <LV_bAxt> }
rule Vbar_3S_bAxt:w    { <AUX_3S> <LV_bAxt> }
rule Vbar_1P_bAxt:w    { <AUX_1P> <LV_bAxt> }
rule Vbar_2P_bAxt:w    { <AUX_2P> <LV_bAxt> }
rule Vbar_3P_bAxt:w    { <AUX_3P> <LV_bAxt> }
rule Vbar_1S_kubid:w   { <AUX_1S> <LV_kubid> }
rule Vbar_2S_kubid:w   { <AUX_2S> <LV_kubid> }
rule Vbar_3S_kubid:w   { <AUX_3S> <LV_kubid> }
rule Vbar_1P_kubid:w   { <AUX_1P> <LV_kubid> }
rule Vbar_2P_kubid:w   { <AUX_2P> <LV_kubid> }
rule Vbar_3P_kubid:w   { <AUX_3P> <LV_kubid> }
rule Vbar_1S_grdid:w   { <AUX_1S> <LV_grdid> }
rule Vbar_2S_grdid:w   { <AUX_2S> <LV_grdid> }
rule Vbar_3S_grdid:w   { <AUX_3S> <LV_grdid> }
rule Vbar_1P_grdid:w   { <AUX_1P> <LV_grdid> }
rule Vbar_2P_grdid:w   { <AUX_2P> <LV_grdid> }
rule Vbar_3P_grdid:w   { <AUX_3P> <LV_grdid> }
rule Vbar_1S_cid:w     { <AUX_1S> <LV_cid> }
```

```
rule Vbar_2S_cid:w     { <AUX_2S> <LV_cid> }
rule Vbar_3S_cid:w     { <AUX_3S> <LV_cid> }
rule Vbar_1P_cid:w     { <AUX_1P> <LV_cid> }
rule Vbar_2P_cid:w     { <AUX_2P> <LV_cid> }
rule Vbar_3P_cid:w     { <AUX_3P> <LV_cid> }
rule Vbar_1S_brid:w    { <AUX_1S> <LV_brid> }
rule Vbar_2S_brid:w    { <AUX_2S> <LV_brid> }
rule Vbar_3S_brid:w    { <AUX_3S> <LV_brid> }
rule Vbar_1P_brid:w    { <AUX_1P> <LV_brid> }
rule Vbar_2P_brid:w    { <AUX_2P> <LV_brid> }
rule Vbar_3P_brid:w    { <AUX_3P> <LV_brid> }
...

# Lexicon
rule NP_1S      { mn }
rule NP_2S      { tu }
rule NP_3S      { u | mACin | uqt | mrd | CxS | ... }
rule NP_1P      { mA }
rule NP_2P      { CmA }
rule NP_3P      { AnhA }

rule NV_krd     { kmk | ElAm | pidA | kAr | sfr | ... }
rule NV_Cd      { uArd | Ab | ... }
rule NV_dAd     { pCt | qrAr | dst | ... }
rule NV_zd      { dst | rqm | Drbh | tup | Hrf | ... }
rule NV_bud     { CAml | ... }
rule NV_dACt    { dust | HDur | dnbAl | pi | qrAr | ... }
rule NV_grft    { tElq    | ... }
rule NV_rft     { qrAul   | ... }
rule NV_kCid    { dst     | ... }
rule NV_andAxt  { dst     | ... }
rule NV_xurd    { Ckst    | ... }
rule NV_gLACt   { qAl     | ... }
rule NV_Aurd    { Eml     | ... }
rule NV_sAxt    { mrbuT   | ... }
rule NV_gft     { mhrmAnh | ... }
rule NV_iAft    { dst     | ... }
rule NV_bst     { cCm     | ... }
rule NV_brd     { nAm     | ... }
rule NV_Amd     { Eml     | ... }
rule NV_rixt    { fru     | ... }
rule NV_oftAd   { etfAq   | ... }
rule NV_nmud    { xnVA    | ... }
rule NV_rsAnd   { ziAn    | ... }
rule NV_brdACt  { dst     | ... }
rule NV_jst     { dl      | ... }
rule NV_bxCid   { ruHih   | ... }
rule NV_xuAnd   { frA     | ... }
rule NV_pidAkrd { Cib     | ... }
rule NV_rsid    { xdmt    | ... }
rule NV_did     { Asib    | ... }
rule NV_bAxt    { jAn     | ... }
rule NV_kubid   { xAl     | ... }
```

```
  rule NV_grdid   { mntj    | ... }
  rule NV_cid     { cCm     | ... }
  rule NV_brid    { omid    | ... }
  ...

  rule AUX_1S   { xuAhm  }
  rule AUX_2S   { xuAhi  }
  rule AUX_3S   { xuAhd  }
  rule AUX_1P   { xuAhim }
  rule AUX_2P   { xuAhid }
  rule AUX_3P   { xuAhnd }

  rule LV_krd     { krd     }
  rule LV_Cd      { Cd      }
  rule LV_dAd     { dAd     }
  rule LV_zd      { zd      }
  rule LV_bud     { bud     }
  rule LV_dACt    { dACt    }
  rule LV_grft    { grft    }
  rule LV_rft     { rft     }
  rule LV_kCid    { kCid    }
  rule LV_andAxt  { andAxt  }
  rule LV_xurd    { xurd    }
  rule LV_gLACt   { gLACt   }
  rule LV_Aurd    { Aurd    }
  rule LV_sAxt    { sAxt    }
  rule LV_gft     { gft     }
  rule LV_iAft    { iAft    }
  rule LV_bst     { bst     }
  rule LV_brd     { brd     }
  rule LV_Amd     { Amd     }
  rule LV_rixt    { rixt    }
  rule LV_oftAd   { oftAd   }
  rule LV_nmud    { nmud    }
  rule LV_rsAnd   { rsAnd   }
  rule LV_brdACt  { brdACt  }
  rule LV_jst     { jst     }
  rule LV_bxCid   { bxCid   }
  rule LV_xuAnd   { xuAnd   }
  rule LV_pidAkrd { pidAkrd }
  rule LV_rsid    { rsid    }
  rule LV_did     { did     }
  rule LV_bAxt    { bAxt    }
  rule LV_kubid   { kubid   }
  rule LV_grdid   { grdid   }
  rule LV_cid     { cid     }
  rule LV_brid    { brid    }
  ...

}


say "Grammatical" if m:w/^ <Persian.sentence> $/;
```

Appendix B

A Perl 6 Grammar for Persian Passive Crossing Dependencies

```
#!/usr/bin/pugs

### Jon Dehdari, 2006
### A Perl 6 context-free grammar to recognize Persian crossing dependencies in passive constructions

use v6;

$_ = shift || "AnhA gCudh xuAhnd Cd";

grammar Persian {
  # Grammatical stuff
  rule sentence  {
      <NP_1S> <vP_1S> | <NP_2S> <vP_2S> | <NP_3S> <vP_3S> |
      <NP_1P> <vP_1P> | <NP_2P> <vP_2P> | <NP_3P> <vP_3P>
  }

  rule vP_1S:w   { <VP_PAS> <vbar_1S_PAS> }
  rule vP_2S:w   { <VP_PAS> <vbar_2S_PAS> }
  rule vP_3S:w   { <VP_PAS> <vbar_3S_PAS> }
  rule vP_1P:w   { <VP_PAS> <vbar_1P_PAS> }
  rule vP_2P:w   { <VP_PAS> <vbar_2P_PAS> }
  rule vP_3P:w   { <VP_PAS> <vbar_3P_PAS> }

  rule vbar_1S_PAS:w      { <AUX_1S> Cd }
  rule vbar_2S_PAS:w      { <AUX_2S> Cd }
  rule vbar_3S_PAS:w      { <AUX_3S> Cd }
  rule vbar_1P_PAS:w      { <AUX_1P> Cd }
  rule vbar_2P_PAS:w      { <AUX_2P> Cd }
  rule vbar_3P_PAS:w      { <AUX_3P> Cd }

  # Lexicon
  rule NP_1S    { mn }
  rule NP_2S    { tu }
```

```
  rule NP_3S    { u | mACin | uqt | mrd | CxS | ... }
  rule NP_1P    { mA }
  rule NP_2P    { CmA }
  rule NP_3P    { AnhA }

  rule AUX_1S   { xuAhm  }
  rule AUX_2S   { xuAhi  }
  rule AUX_3S   { xuAhd  }
  rule AUX_1P   { xuAhim }
  rule AUX_2P   { xuAhid }
  rule AUX_3P   { xuAhnd }

  rule VP_PAS   { gCudh | dAdh | gLACth | grfth | sAxth | kCidh | brdACth | zdh | ... }
}


say "Grammatical" if m:w/^ <Persian.sentence> $/;
```

Appendix C

Romanization and Transliteration

Persian is normally written in the Perso-Arabic script, an extension of the Arabic script. It is written from right-to-left, and omits three vowels: /æ e o/ . All of the Persian words and sentences in this thesis are written using the homomorphic romanization scheme listed in the third column of the following table. The Perl scripts in Appendices A and B use the monomorphic transliteration found in the second column. Most modern digital texts in the Perso-Arabic script are encoded in UTF-8, CP-1256, ISIRI 3342, or HTML numeric character references.

Perso-Arabic Script	Dehdari translit.	Romanization	ArabTeX	Uni-Dec	Uni-Hex	UTF-8	ISIRI 3342	CP1256	Unicode Name
ا	A	ā/æ/o/e	A	1575	0627	d8a7	c1	c7	ARABIC LETTER ALEF
ب	b	b	b	1576	0628	d8a8	c3	c8	ARABIC LETTER BEH
پ	p	p	p	1662	067e	d9be	c4	81	ARABIC LETTER PEH
ت	t	t	t	1578	062a	d8aa	c5	ca	ARABIC LETTER TEH
ث	V	s	_t	1579	062b	d8ab	c6	cb	ARABIC LETTER THEH
ج	j	j	j	1580	062c	d8ac	c7	cc	ARABIC LETTER JEEM
چ	c	č	^c	1670	0686	da86	c8	8d	ARABIC LETTER TCHEH
ح	H	h	.h	1581	062d	d8ad	c9	cd	ARABIC LETTER HAH
خ	x	x	x	1582	062e	d8ae	ca	ce	ARABIC LETTER KHAH
د	d	d	d	1583	062f	d8af	cb	cf	ARABIC LETTER DAL
ذ	L	z	_d	1584	0630	d8b0	cc	d0	ARABIC LETTER THAL
ر	r	r	r	1585	0631	d8b1	cd	d1	ARABIC LETTER REH
ز	z	z	z	1586	0632	d8b2	ce	d2	ARABIC LETTER ZAIN
ژ	J	ž	^z	1688	0698	da98	cf	8e	ARABIC LETTER JEH
س	s	s	s	1587	0633	d8b3	d0	d3	ARABIC LETTER SEEN
ش	C	š	^s	1588	0634	d8b4	d1	d4	ARABIC LETTER SHEEN
ص	S	s	.s	1589	0635	d8b5	d2	d5	ARABIC LETTER SAD
ض	D	z	.d	1590	0636	d8b6	d3	d6	ARABIC LETTER DAD
ط	T	t	.t	1591	0637	d8b7	d4	d8	ARABIC LETTER TAH
ظ	Z	z	.z	1592	0638	d8b8	d5	d9	ARABIC LETTER ZAH
ع	E	ʼ	ʻ	1593	0639	d8b9	d6	da	ARABIC LETTER AIN
غ	G	q/ġ	.g	1594	063a	d8ba	d7	db	ARABIC LETTER GHAIN
ف	f	f	f	1601	0641	d981	d8	dd	ARABIC LETTER FEH
ق	q	q/ġ	q	1602	0642	d982	d9	de	ARABIC LETTER QAF
ک	k	k	k	1705	06a9	daa9	da	98	ARABIC LETTER KEHEH
گ	g	g	g	1711	06af	daaf	db	90	ARABIC LETTER GAF
ل	l	l	l	1604	0644	d984	dc	e1	ARABIC LETTER LAM
م	m	m	m	1605	0645	d985	dd	e3	ARABIC LETTER MEEM
ن	n	n	n	1606	0646	d986	de	e4	ARABIC LETTER NOON
و	u	u/v/w	U	1608	0648	d988	df	e6	ARABIC LETTER WAW
ه	h	h	h	1607	0647	d987	e0	e5	ARABIC LETTER HEH
ی	i	i/y	I	1740	06cc	db8c	e1		ARABIC LETTER FARSI YEH
ـَ	a	æ	a	1614	064e	d98e	f0	f3	ARABIC FATHA
ـُ	o	o	o	1615	064f	d98f	f2	f5	ARABIC DAMMA
ـِ	e	e	e	1616	0650	d990	f1	f6	ARABIC KASRA
آ	]	ā	ʼA	1570	0622	d8a2	c0	c2	AR. LET. ALEF WITH MADDA ABOVE
ء	M	ʼ	ʼ|	1569	0621	d8a1	c2	c1	ARABIC LETTER HAMZA
ۀ	X	eye	H-i	1728	06c0	db80		c0	AR. LET. HEH WITH YEH ABOVE
ئ	I	ʼi	ʼy	1574	0626	d8a6	fb	c6	AR. LET. YEH WITH HAMZA ABOVE
ؤ	U	oʼ	Uʼ	1572	0624	d8a4	fa	c4	AR. LET. WAW WITH HAMZA ABOVE
اً	N	æn	aN	1611	064b	d98b	f3	f0	ARABIC FATHATAN
ـّ	~	*xx*	*xx*	1617	0651	d991	f6	f8	ARABIC SHADDA
»	{	“	`\lq`	0171	00ab	ab	e7	ab	LEFT-POINTING DOUBLE ANGLE …
«	}	”	`\rq`	0187	00bb	bb	e6	bb	RIGHT-POINTING DOUBLE ANGLE …
	-	-	`\hspace{0ex}`	8204	200c	e2808c	a1	9d	ZERO WIDTH NON-JOINER