Charles University in Prague

Faculty of Mathematics and Physics

M.Sc. THESIS

Lasha Abzianidze

An HPSG-based Formal Grammar of a Core Fragment of Georgian Implemented in TRALE

Institute of Formal and Applied Linguistics

Supervisor: Ing. Alexandr Rosen Ph.D.
Co-Supervisor: Prof. Patrick Blackburn

Study program: European Masters Program in Language and Communication Technology (LCT)

Prague 2011

Název práce: Formální gramatika jádra gruzínštiny
podle teorie HPSG, implementovaná v systému TRALE
Autor: Lasha Abzianidze
Katedra: Ústav formální a aplikované lingvistiky,
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze.
Vedoucí diplomové práce: Ing. Alexandr Rosen Ph.D.,
Ústav teoretické a komputační lingvistiky,
Filozofická fakulta, Univerzita Karlova v Praze.
Abstrakt: Gruzínština se výrazně odlišuje od indoevropských jazyků. Vyznačuje se řadou jazykových jevů, které jsou obtížné pro lingvistickou teorii i počítačové zpracování. Navíc patří mezi jazyky, u kterých nelze plně využít existující zdroje, a není ani dostatečně prozkoumána z hlediska matematické lingvistiky. Cílem této práce je vytvořit formální gramatiku morfologie a syntaxe jádra gruzínštiny. Tato formální gramatika vychází z teorie HPSG, která je v současnosti jedním z nejúspěšnějších rámců pro formální popis jazyka. Gramatiku implementujeme v systému TRALE, který umožňuje věrné zachycení ručně psaných gramatik založených na HPSG. Tato práce je první aplikací teorie HPSG na gruzínštinu.
Klíčová slova: gruzínština, HPSG, TRALE, formální gramatika, komplementace, adjunkce, lexikální pravidla, logická deklinační paradigmata, polypersonální konjugační paradigmata

Title: An HPSG-based formal grammar of
a core fragment of Georgian implemented in TRALE
Author: Lasha Abzianidze
Department: Institute of Formal and Applied Linguistics,
Faculty of Mathematics and Physics, Charles University in Prague.
Supervisor: Ing. Alexandr Rosen Ph.D.,
Institute of Theoretical and Computational Linguistics,
Faculty of Arts, Charles University in Prague.
Abstract: Georgian is remarkably different from Indo-European languages. The language has several linguistic phenomena that are challenging both from theoretical and computational points of view. In addition, it is low-resourced and insufficiently studied from the computational point of view. In the thesis, we model morphology and syntax of a core fragment of the language in a formal grammar. Namely, the formal grammar is written in the HPSG framework – one of the most powerful grammar framework nowadays. We also implement the grammar in TRALE – a grammar implementation platform, which is faithful to "hand-written" HPSG-based grammars. Note that this is the first application of HPSG to Georgian.
Keywords: Georgian, HPSG, TRALE, formal grammar, complementation, adjunction, lexical rules, logical declension paradigms, polypersonal conjugation paradigms

Contents

List of Figures

List of Tables

List of Codes

1. Introduction to Georgian and its Grammar

In this chapter, we shortly introduce the Georgian language. As the language has its own script, we show its IPA chart along with two transliteration systems, and also explain conventions used in glosses. Then we shift to the Georgian grammar and discuss its basic properties and interesting phenomena, list Georgian-specific challenges for computational linguistics and briefly overview research done on the language so far. Finally, we characterize the current position of the language from the point of view of computational linguistics.

1.1 The Georgian language and its script

The Georgian language is a native language for Georgians and the official language of Georgia. It has about 5 million native speakers. The language belongs to the Kartvelian (also know as South Caucasian) language family, which is indigenous for the Caucasus – the region between Europe and Asia. Note that there is no known relation between the Kartvelian and other language families.

Georgian uses its own unique alphabet for writing. It is called Mkhedruli ("cavalry" or "military") and consists of 33 letters: 28 consonants and 5 vowels. The direction of reading and writing is from the left to the right and the reading rules are trivial – phonemes and graphemes are into one-to-one correspondence.

Geo	IPA	Tra	Lat	Geo	IPA	Tra	Lat	Geo	IPA	Tra	Lat
ა	ɑ	a	a	მ	m	m	m	ღ	ɣ	ḡ	gh
ბ	b	b	b	ნ	n	n	n	ყ	q'	q	y
გ	g	g	g	ო	o	o	o	შ	ʃ	š	sh
დ	d	d	d	პ	p'	p̣	p	ჩ	$tʃ^h$	č	ch
ე	ɛ	e	e	ჟ	ʒ	ž	zh	ც	ts^h	c	c
ვ	v	v	v	რ	r	r	r	ძ	dz	ǰ	dz
ზ	z	z	z	ს	s	s	s	წ	ts'	c̣	ts
თ	t^h	t	th	ტ	t'	ṭ	t	ჭ	tʃ'	č̣	tch
ი	i	i	i	უ	u	u	u	ხ	x	x	x
კ	k'	ḳ	k	ფ	p^h	p	f	ჯ	dʒ	j	j
ლ	l	l	l	ქ	k^h	k	q	ჰ	h	h	h

Table 1.1: displays the IPA and two transliterations: a letter-to-letter and a letter-to-sequence transliterations.

The table 1.1 shows characters of the Georgian script and their representation in the international phonetic alphabet (IPA), and by a letter-to-letter and a letter-to-sequence transliterations. This is the first and last time we are writing these "weird and round" Georgian letters – throughout this work Georgian will be written with the letter-to-letter transliteration. The letter-to-sequence transliteration will be used in the source code of the implemented formal grammar in order to facilitate the implementation of the code.

While using examples of Georgian words or sentences, we will write analytical glosses of the expression in English below. What we mean by the analytical gloss is that words or sentences will be rendered word by word, preserving their order and (word boundary) spaces between words. On the other hand, the word-to-word translation will be done morpheme by morpheme in the following way: if the translation of a morphemes is available in English, the morpheme will be translated, otherwise it will be shown as abbreviation of the grammatical feature expressed by this morpheme. Everything that is a part of the translation of a certain word will be connected by hyphens like beads on a string. Often words in Georgian correspond to sentences in English – wherever a strict morpheme-to-morpheme translation will be unnecessary or complex (requiring details irrelevant for the discussion) we will try to translate it using English words. Although the morpheme-to-morpheme translation will not be prefect, the composition of such translation will be for the most part identical to the Georgian word. Also, when the order of morphemes is not relevant we will ignore it. All these simplifications will be done without missing anything important for the discussion and for the reader. If the example includes a sentence, we will write the most adequate translation of the sentence in English below the analytical gloss.

1.2 Introduction to the Georgian grammar

In this section we are going to briefly characterize the Georgian grammar, to underline its significant syntactic and morphological properties. Why briefly? Because we will introduce the grammar in the standard manner, as it is commonly done by both computational and non-computational linguists working on Georgian. In the next chapters, where we will be constructing the formal grammar, some points of the grammar will be revisited, providing arguments to agree or disagree with the standard views on these points.

The first thing to note about the language – it is considered to be an agglutinative language, which means that morphemes are simply concatenated (or "glued") with roots:

(1) *sa-avad-mqop-o-eb-ši*
for-ill-being-PL-in
In hospitals

In (1), several instances of "gluing" of morphemes in the Georgian word are exemplified. Each "gluing" has its position in the process: first, the stem[1] *avad-mqop* (ill[2]) is created by compounding the words *avad* (ill) and *mqop*[3] (being). Then the word *sa-avadmqop-o* 'hospital' is derived by the affixation of the stem

[1]As the notion of *stem* is often used with slightly different meanings, we specify its meaning. In this work, *stem* will be understood as the part of the word that is common to all inflected variants of the word.

[2]In English, *ill* is not understood as a noun referring to smb./smth. who/what is ill. On the other hand, Georgian *avad-mqop-i* can be understood as a noun or an adjective. At this moment the nature of the word is not relevant and will be discussed in the later chapters.

[3]In order to simplify things, we do not go further and analyze the morphology of the words *avad* and *mqop*.

avadmqop with the circumfix *sa- -o*, which – after its application – derives a new lexical word (hospital), denoting something which is intended *for* things (e.g. ill people) denoted by the original word (ill). Then the inflection, namely pluralization, is applied by “gluing” the plural suffix *-eb* to the stem *saavadmqopo*[4] and we got the stem *saavadmqopo-eb*. In the end, the suffix, namely postposition *-ši* ‘in’ is “glued” to the stem *saavadmqopoeb*. The process of “gluing” is shown by the parenthesis in (2):

(2) *((sa-(avad-mqop)-o)-eb)-ši*

In addition to the simple “gluing” property of the Georgian morphology, we read the following in the Wikipedia’s topic *Agglutinative language*:

“Agglutinative languages tend to have a high rate of affixes/morphemes per word, and to be very regular[citation needed]. ... Georgian is an exception; not only is it highly agglutinative (there can be simultaneously up to 8 morphemes per word), but there is also a significant number of irregular verbs, varying in degrees of irregularity.”

Moreover, some linguists consider the Georgian language to be an inflectional-agglutinative language (Butskhrikidze, 2002), having both inflectional and agglutinative properties. To continue talking about the Georgian morphology, we will characterize the morphology of the main word classes (i.e. parts-of-speech).

Traditionally, Georgian nouns are assumed to have 7 cases: nominative, ergative, dative, genitive, instrumental, adverbial and vocative. Cases are marked with suffixes, but at the same time, noun suffixes can express postpositions. So, distinguishing the case inflection from the postpositional inflection is a tricky part of Georgian grammar.[5] Similarly to nouns, adjectives also have cases. Georgian adjectives can stand alone or modify nouns in the sentence, so depending on their status they have fewer number of cases (we can also look at this fact in the way that the number of cases is the same, but some cases exhibit syncretic forms). The same can be said about the case inflections of quantifiers. It is also possible to decline pronouns and non-finite verbs by case.

The syncope and apocope (the loss of sounds in the middle or at the end of the word respectively) are common phenomena during declension and pluralization. In (3) and (4), there is an example of both syncope (the loss of *e*) and apocope (the loss of *a*) at the same time in the word stem. When it is important, we encode the lost letters in the superscript:

(3) *p̣ep̣ela*
butterfly-NOM

(4) *p̣ep̣*^{e}*l*^{a}*-eb-i*

4In general, there are many derived nouns in Georgian and their morpheme by morpheme translation will be often irrelevant for some discussions, so we will simply translate the referent with the grammatical features. E.g. in case of *saavadmqopo* we will simply write *hospital*-NOM in the analytical gloss.

5Moreover, there were/are several discussions about the number of cases. In addition to these 7 standard cases some linguists consider other noun inflections as case inflections and some oppose the standard list of cases by arguing that the particular inflection is not a case inflection. We will revisit this issue when we start building a formal grammar for Georgian.

butterflies-NOM

There are also some *irregular* alternations occurring rarely in the word stems, they will be explored in the later chapters.

Now it is time to introduce the verb – the key word class in the Georgian language. What is special to the Georgian verb? The verb often serves as a “backbone” (the fundamental component to which other components are attached) of a sentence from the semantic and morphosyntactic points of view. The former view is the common one in most (probably in all) languages, but the latter one is not quite obvious and we are going to demonstrate it now.

The point is that Georgian verbs are able to encode the information about their subjects, objects, indirect objects,[6] tense, aspect, mood, and direction of the action in their morphology. The best way to understand this fact is to give examples:

(5) *ča-g-i-qvan-e*
down-2SING-take-1SING-PST-IND-PFV
I took you down to

(6) *ča-g-e-qvan-e*
down-1SING-take-2SING-PRS-SJV-IPFV
If you took me down to

In (5), the prefix *ča-* is called a directional preverb and shows the direction of the action or state expressed by the verb. Note that it is difficult to translate the verb morpheme by morpheme[7] as some morphemes often give specific information in combination with other morphemes. For example, (6) shows how the change of one morpheme has reversed the subject-object pair and totally changed the tense-aspect-mood (TAM).

(7) *m-a-dar-eb*
1SING-compare-2SING-3-PRS-IND-IPFV
You are comparing me to her[8]/them

(8) *m-a-dar-eb-s*
1SING-compare-3SING-3-PRS-IND-IPFV
She is comparing me to her/them

Moreover, the omission of the morpheme can serve as a marker of a different (opposite to what is expressed by the omitted morpheme) grammatical category, as there are some grammatical categories (e.g. in some cases 2nd person singular) without corresponding morpheme in certain environments. This fact is demonstrated in (7) and (8), when the 3rd person morpheme *-s* is omitted.

(9) *mo-g-a-bar-eb-d-e-t*
to-2PL-hand-1SING-3-FUT-SJV-PFV ;

[6]At this point we are using common notions of syntactic functions, but as we will see later these notions will not be as relevant as they are e.g. in Indo-European languages.

[7]So, the glosses are not flawless but they are organized in a way that they show the discussed phenomena adequately.

[8]Since there is no gender in the Georgian grammar, the reader should ignore the information about gender provided by the English personal pronouns in English translations.

to-2PL-hand-1PL-3-FUT-SJV-PFV ;
to-2SING-hand-1PL-3-FUT-SJV-PFV
I/We would hand you her/them

Often the number of the subject, object or indirect object is not explicitly expressed in the morphology of the Georgian verb. The reasons are the following: non-existence of the specific morpheme in certain environments (for indirect object in (7) and (8)), or the coincidence of the morphemes of different grammatical functions (in (9), plural morpheme *-t* can express the number of the subject and the object).

One of the features of the verb we have just demonstrated is called polypersonal agreement. The morphological feature of the verb to express (*agree* with) more than one of the *arguments*[9] is called *polypersonal agreement* or *polypersonalism.* Polypersonal agreement in actual sentences allows the omission of most pronouns without losing any information (demonstrated in (10) and (11)), in other words the pronoun-dropping is allowed and this fact categorizes Georgian as the *pro-drop* language.

(10)	*me*	*mo-g-a-bar-eb-d-e*	*šen*
	1SING-NOM	PVB-2SING-hand-1SING-3-FUT-SJV-PFV	2SING-DAT
	mat		
	3PL-DAT		
	I would hand you them		

(11)	*mo-g-a-bar-eb-d-e*	*mat*
	PVB-2SING-hand-1SING-3-FUT-SJV-PFV	3PL-DAT
	I would hand you them	

While talking about TAM grammatical features of the verb, we have to mention their specific representation suited for the Georgian verb. Traditionally the Georgian grammar uses the notion of *row* (*mc̣ḳrivi*, adopted in English as *screeve*) to describe the verbs according to the TAM feature. There are 11 screeves assumed in the traditional grammar. Each screeve represents the set of 6 word forms of the verb.[10] Table 1.2 shows how these 11 screeves are grouped in series and what the possible values of the TAM grammatical features are.

Note that Series I is divided into two subgroups – present and future. Linguists often include the 12th screeve in Series II, which stands for the imperative screeve with the imperative mood. Also, there were attempts by some linguists to argue for decreasing the number of screeves to 8 or increasing their number up to 16 based on some views.

It is common in Georgian that some verbs cannot have corresponding word forms in some screeves or even in some series. There are different rules of word-formation for different kinds of verbs inside the screeves. The latter fact makes it difficult to guess what kind of inflection a certain verb will have in a certain

[9]By *argument* we mean a phrase (or a word) which is in a syntactic relation with the verb.

[10]Why 6 word forms? Because of the argument of the verb, there are 6 possible combinations of grammatical features of person and number. However, we have seen a polypersonal agreement feature of the Georgian verb, which implies that some verbs can agree with more than one argument, therefore sometimes it should have more than 6 word forms. This is a reasonable point for discussion and it will be discussed in the later chapters.

Series	Screeves	Tense	Mood	Aspect
I	Present	Present (Future, General)	IND	IPFV
	Imperfect	Past	IND	
	Present Subjunctive	Present (Future)	SJV	
	Future	Future	IND	PFV
	Conditional	Past	IND	
	Future Subjunctive	Future	SJV	
II	Aorist	Past	IND	(I)PFV
	Optative	Present / Future	SJV	
III	Perfect	Past	IND	
	Pluperfect	Past	IND	
	Perfect Subjunctive	Past / Future	SJV	

Table 1.2: Traditional 11 screeves grouped in 3 series. For each screeve, possible values of the TAM grammatical features are indicated.

screeve. Moreover, different kind of verbs are conjugated differently inside some screeves.

Georgian verbs are traditionally divided into four classes. The division is based on features such as: having word forms for a certain set of screeves or series, how the stems of different series are related to each other and the transitive or intransitive property of the verb. The verbs in the same classes are supposed to conjugate similarly according to number and person or to screeves. Unfortunately, there are many *irregular* verbs which do not fall in any of these classes. The following citation from Wikipedia emphasizes the same problem:

"Georgian verb conjugation remains a tough subject even for the people who have been studying the language for a while. Even after studying over hundreds of verbs, one may still encounter a new verb whose conjugation deviates from what the person has learnt. This is not to say that the verbs are irregular, rather, to state that verbs in Georgian do not tend to conform to a 'universal' conjugation system like in most other languages."

The Georgian verb is a good example of morphology interfacing with syntax. As we have mentioned, the verb already encodes the information about syntactic functions of its arguments – subject, object and indirect object – in its morphology. Moreover, the verb *governs*[11] these syntactic arguments by marking them with grammatical cases. In traditional Georgian grammar, it is assumed that subjects, objects and indirect objects are marked with three cases: nominative, ergative[12] and dative. Here, we show some examples of case alignment in sentences:

[11]For readers who are not familiar with syntactic relations, governing is used to express the specific relation between two word class (or phrases) – when a (main or so-called head) word marks another (dependent) word with some grammatical features, which the former does not show. E.g. prepositions mark nouns with cases, or verbs mark nouns with case, though prepositions and verbs do not show the case feature.

[12]This grammatical case is sometimes referred to as narrative or aorist case. In general, the ergative case is used to mark subject or agent of transitive verbs.

(12) *sṭudenṭ-i ḳitx-ul-ob-s ċign-s*
student-NOM reads-PRS-IPFV-1SING book-DAT
The student reads / is reading the book

(13) *sṭudenṭ-ma ċa-i-ḳitx-a ċign-i*
student-ERG PVB-read-PST-PFV-1SING book-NOM
The student read the book

In (12), there is nominative-dative case alignment in present (screeve 1) while in (13), for the lexically identical verb the case alignment has changed to ergative-nominative in Aorist (screeve 7). This kind of behavior is characteristic to most of so-called *dynamic* verbs – verbs whose subject is the agent of the event expressed by the verb.

(14) *ǰaḡl-i mi-h-qv-eb-a paṭron-s*
dog-NOM PVB-3-follow-PRS-textscipfv-3SING master-DAT
The dog follows the master

(15) *ǰaḡl-i ga-h-qv-a paṭron-s*
dog-NOM PVB-3-follow-PST-PFV-3SING master-DAT
The dog followed the master

But in (14) and (15), the alignment nominative-dative remains constant with respect to screeve 1 and screeve 7 (indeed to all screeves). There is one more interesting alignment in (16) and (17), where the traditional subject of the verb is marked with dative case and the traditional object with nominative case, unlike in the Indo-European languages. Note that the above-mentioned alignment also holds in all screeves.

(16) *ǰaḡl-s u-qvar-s paṭron-i*
dog-DAT 3-love-PRS-textscipfv-3SING master-NOM
The dog loves the master

(17) *ǰaḡl-s e-qvar-eb-a paṭron-i*
dog-DAT 3-love-FUT-PFV-3SING master-NOM
The dog will love the master

Based on the discussed examples, the language is considered as having the split ergativity property meaning that the language partially (un)satisfies the ergative-absolutive property.[13]

In Georgian, case alignment is defined mainly by the lexical semantics of the verb rather than by other information. In addition, application of a certain pre-verb to the verb is also guided by lexical semantics of the verb. In cases where the verb expresses action having a direction feature (e.g. to go, to send, to hand, etc.), the applicable preverbs can serve as directional markers, otherwise they serve as tense and aspect markers. Therefore we believe that lexical semantics of Georgian verbs are the main reason for all *irregularities* related to their conjugation according to person-number and screeves.

[13]The property says that the language maintains a syntactic or morphological equivalence (such as the same word order or grammatical case) for the object of a transitive verb and the single core argument of an intransitive verb, while treating the agent of a transitive verb differently [Wikipedia].

As the verb agrees with and governs subjects, objects and indirect objects, there is no need to recode the same information by word order – the language is considered a *free word order* language at the *phrase* level. We show this property on the example (18), where any subject-verb-object order is acceptable[14].

(18) *deda-m bavšv-s saṭamašo u-qid-a*
Mother-ERG child-DAT toy-NOM 3-buy-3SING-3-PST-PFV
The mother bought the toy to the child

(19) *deda-m bavšv-s saṭamašo u-qid-a*
deda-m u-qid-a saṭamašo bavšv-s
saṭamašo bavšv-s deda-m u-qid-a
bavšv-s saṭamašo u-qid-a deda-m

In (19), we give some licensed word orders in the sentence. The free word order property will be revisited during modeling the Georgian syntax.

Above we emphasized the free word order on the phrase level – in general the word order is fixed inside the phrases. In noun phrases, adjectives precede[15] nouns. The same kind of precedence exists between determiners (demonstratives, possessive determiners, or quantifiers) and nouns. The noun phrase has the property of *left-branching*. The example of left-branching is shown in (20):

(20) *čem-i megob*^{a}*r-is col-is mankan-is gasaḡeb-i*
my-NOM friend-GEN wife-GEN car-GEN key-NOM
The key of the car of the wife of my friend

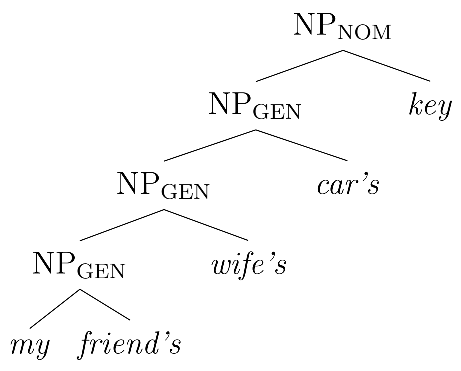


Figure 1.1: Left-branching in the noun phrase

In figure 1.1, a simplified parse tree demonstrates the left-branching property. In the top-to-bottom direction, the tree is "growing" on the left hand side. The head of the noun phrase (*key*) is the first right-branch of the tree.

In noun phrases, adjectives and possessive pronouns agree with heads only in case, while quantifiers apply to the noun phrase whose head is only in singular. Adjectives cannot modify the noun after determiners are applied to it, hence adjectives following determiners is the only licensed word order in noun phrases.

[14]Maybe some orders are rarely used, hence traditionally some of them are considered ungrammatical but in the examples all orders make clear sense.

[15]The precedence of nouns to adjectives is not common for Georgian, but this phenomenon is observed in Georgian lyrics. In the thesis, we are only considering the common case.

There is no grammatical gender distinction in Georgian, there is even just one kind of pronoun referring to the 3rd person (unlike in English), but there is the animacy feature, categorizing nouns on whether their referents are animate or inanimate. The animacy grammatical feature plays a role on the syntax level. While nouns agree with verbs in the number feature, inanimate nouns cannot agree with verbs in plural number, but animate ones can:

(21) *adamian-eb-s s-č̣ir-d-eb-a-t c̣qal-i*
Humans-DAT 3-need-PRS-IPFV-3PL water-NOM
Humans need water

Example (21) shows the licensed agreement in number between the plural animate(!) subject (*humans*) and the verb with the plural(!) subject marker. On the other hand, (22) shows the licensed agreement in number between the plural inanimate(!) subject (*trees*) and the verb with the singular(!) subject marker. If we substitute the plural feature of the subject for the singular feature the sentence (23) will still remain grammatically correct.

(22) *xe-eb-s s-č̣ir-d-eb-a c̣qal-i*
Trees-DAT 3-need-PRS-IPFV-3SING water-NOM
Trees need water

(23) *xe-s s-č̣ir-d-eb-a c̣qal-i*
Tree-DAT 3-need-PRS-IPFV-3SING water-NOM
The tree needs water

(24) **xe-eb-s s-č̣ir-d-eb-a-t c̣qal-i*
Trees-DAT 3-need-PRS-IPFV-3PL water-NOM

But (24) example is not grammatically correct because inanimate subject cannot agree with the verb with the plural(!) subject marker[16]. According to this examples, one can say that there is no syntactic distinction between inanimate nouns with the singular and the plural number features, but this is not true. There are some verbs in Georgian which change their stems according to the number feature of the grammatical function governed by them. We use examples to demonstrate this phenomenon:

(25) *saṭamašo gd-i-a magida-ze*
toy-NOM lie-PRS-IPFV-3SING table-DAT[17]-on
The toy lies on the table

(26) **saṭamašo-eb-i gd-i-a magida-ze*
toys-NOM lie-PRS-IPFV-3SING table-DAT-on

As it is shown in (25) and (26), the substitution of the singular number of the subject for the plural number makes the sentence ungrammatical. The sentence (27) will be grammatical again if we substitute the verb *gd-i-a* for its semantically almost equivalent verb *qr-i-a*. The difference between these verbs is that the

[16]Though, some authors treat inanimate objects as animate. Therefore if we considered the tree as animate object, then (24) and (23), but (22), would be grammatical sentences.

[17]In traditional Georgian grammar, it is assumed that the postposition *-ze* (*in*) governs the noun in dative case and at the same time dative suffix *-s* is lost due to the apocope.

former one governs the subject in singular and the later in plural, there are no other differences on the syntactic or semantic level between them.

(27)	*saṭamašo-eb-i*	*qr-i-a*	*magida-ze*
	toys-NOM	lie-PRS-IPFV-3PL	table-DAT-on
	Toys lie on the table		

We tried to give an introduction to the basic word classes and their grammatical categories in Georgian, and briefly present their morphological and morphosyntactic features. Then we discussed Georgian syntax and showed on examples some of its important properties, such as polypersonal agreement, case alignment (split-ergativity), free word order on the phrase level and fixed word order inside the phrase (left-branching). The reader can find more information about the Georgian language and its (traditional) grammar in English at Wikipedia[18] and (Hillery, 2006).

[18]The information about the language and its grammar at `http://en.wikipedia.org/wiki/Georgian_grammar` and about the verbs at `http://en.wikipedia.org/wiki/Georgian_verb_paradigm`.

2. HPSG Formalism and Its Implementation in TRALE System

We introduce the HPSG formalism, which will be our framework for a formal description of Georgian grammar. In our opinion, the best start for the introduction to the HPSG formalism is a formal one with suitable examples.

First, we introduce the basic elements of the formalism – typed feature structures. We start with a formal introduction to the (untyped) feature structures and provide several examples. Then we introduce the type hierarchy and the signature. Along with the formal definitions and properties, we give illustrative examples of the type hierarchy and the signature. After presenting the points above, we are ready to define typed feature structures over the signature. Feature structures are also illustrated by examples. Next we give formal definitions for the relations of subsumption and equivalence – alphabetic variation, over the typed feature structures. We give a concise and intuitive definition for the abstract typed feature structures and introduce the operation of unification employing those structures.

We start our introduction to the HPSG formalism with a list of its main principles along with their brief explanations. We proceed with a description of components of the formalism: signature, lexicon, lexical rules, principles and grammar rules.

At the end of the chapter, we present TRALE – an implementation platform for HPSG-based grammars. We briefly introduce the architecture of a TRALE grammar and give implemented samples of some components of the HPSG formalism. In this way, we try to make it easy for the reader to read the source code of a simple TRALE grammar.

2.1 Typed feature structures

We introduce typed feature structures, which play crucial role in modeling of linguistic objects in the HPSG formalism. First, we introduce the notion of feature structures and several equivalent views on it; then the notions of type hierarchy and signature will be discussed, followed with typed feature structures. In the end, subsumption order and the operation of unification will be introduced.

2.1.1 Feature structures

Feature structure is a broad concept and – as the name suggests – it rests on the notion of feature. Feature structures are used in many theories and application. The idea behind them is to model any object in terms of a set of features and their corresponding values (*feature-values*), where the value of the feature (i.e. some object) is also represented as a set of feature-value pairs. As you can see feature structures have a recursive structure.

There are different views on feature structures, such as feature graphs (aka concrete feature structures), feature structures, attribute-value matrices or abstract feature structures. But in the thesis, we will work only with feature structures viewed as attribute-value matrices, as this representation is originally used in the HPSG formalism and is common in linguistics. On the other hand, representation of feature structures as graphs is attractive from the theoretical point of view, as it is easy to give formal definitions of the feature structure and its properties in terms of theoretically well studied mathematical objects.

Below, we give definitions for the *unordered signature*[1] and *untyped* feature structures (simply, feature structures) according to (Carpenter, 1993).

Definition: the *unordered signature* Σ is a pair of non-empty finite sets:

$$\Sigma = \langle \mathcal{A}, \mathcal{F} \rangle$$

where $\mathcal{A}$ is a set of atoms and $\mathcal{F}$ is a set of features.

The signature simply represents a fixed set of several kinds of symbols, which is used in formal definitions.[2]

Definition: With respect to the unordered signature $\Sigma = \langle \mathcal{A}, \mathcal{F} \rangle$, a feature graph $\langle Q, q_0, \delta, \theta \rangle$ is a finite, rooted, directed, connected, labeled graph:

Q ($Q \cap \mathcal{A} = Q \cap \mathcal{A} = \emptyset$) is a finite set of nodes,

$q_0 \in Q$ is the root node,

$\delta : Q \times \mathcal{F} \to Q$ is a partial function such that for any $q' \in Q$, there exists a sequences of nodes $\langle q_0, q_1, \ldots, q_n, q' \rangle$ and a sequence of features $\langle f_1, \ldots, f_n \rangle$ that $\delta(q_i, f_{i+1}) = q_{i+1}, (i = 0, \ldots n)$, where $q_{n+1} = q'$,

$\theta : Q_{\mathcal{A}} \to \mathcal{A}$ is a total[3] function, $Q_{\mathcal{A}} = \{q \in Q \mid \delta(q, f) \uparrow \text{ for any } f \in \mathcal{F}\}$.

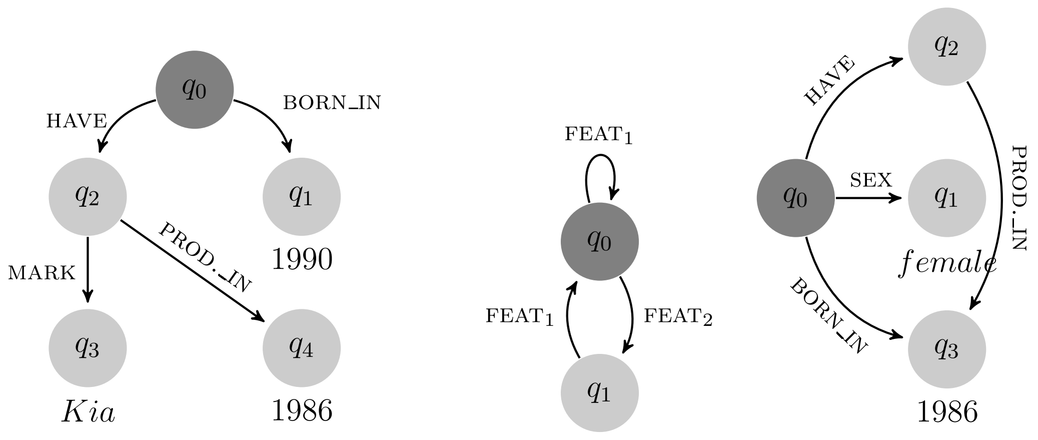
Simply saying, for any node δ function defines its outgoing arcs (features) and their destination nodes. The condition on the δ function requires that any node of the feature graph is reachable from the root node. At the same time θ function assigns labels to the sink nodes (nodes without outgoing arcs). For better under-

[1]We conventionally call it *unordered*, to distinguish it from another kind of signature, which will be introduced later.

[2]Strictly speaking, a signature contains the non-logical symbols of a formal language.

[3]Sometimes assumed as a partial function which allows some features without any values

standing, in Fig.2.1 we give some examples of feature graphs with corresponding parameters:



(a) (b) (c)

(a)
$Q = \{q_0, q_1, q_2, q_3, q_4\}$,
$\langle q_0, \textsc{have}\rangle \overset{\delta}{\mapsto} q_2, \langle q_2, \textsc{mark}\rangle \overset{\delta}{\mapsto} q_3$,
$\langle q_0, \textsc{born\_in}\rangle \overset{\delta}{\mapsto} q_1$,
$\langle q_2, \textsc{prod.\_in}\rangle \overset{\delta}{\mapsto} q_4$,
$q_1 \overset{\theta}{\mapsto} 1990, q_3 \overset{\theta}{\mapsto} \mathit{Kia}, q_4 \overset{\theta}{\mapsto} 1986$.

(b)
$Q = \{q_0, q_1\}$,
$\langle q_0, \textsc{feat}_1\rangle \overset{\delta}{\mapsto} q_0$,
$\langle q_0, \textsc{feat}_2\rangle \overset{\delta}{\mapsto} q_1$,
$\langle q_1, \textsc{feat}_1\rangle \overset{\delta}{\mapsto} q_0$,
$\theta : \emptyset \to \mathcal{A}$.

(c)
$Q = \{q_0, q_1, q_2, q_3\}$,
$\langle q_0, \textsc{have}\rangle \overset{\delta}{\mapsto} q_2, \langle q_0, \textsc{sex}\rangle \overset{\delta}{\mapsto} q_1$,
$\langle q_0, \textsc{born\_in}\rangle \overset{\delta}{\mapsto} q_3$,
$\langle q_2, \textsc{prod.\_in}\rangle \overset{\delta}{\mapsto} q_3$,
$q_1 \overset{\theta}{\mapsto} \mathit{female}, q_3 \overset{\theta}{\mapsto} 1986$.

Figure 2.1: Examples of feature graphs[5] with the unordered signature $\Sigma = \langle \mathcal{A}, \mathcal{F}\rangle$, where $\mathcal{A} = \{1996, 1990, \mathit{female}, \mathit{Kia}\}$ and $\mathcal{F} = \{\textsc{feat}_1, \textsc{feat}_2, \textsc{born\_in}, \textsc{have}, \textsc{mark}, \textsc{prod.\_in}, \textsc{sex}\}$

In Fig.2.1, the root nodes are in darker colors. In feature graphs, it is allowed ingoing arcs in the root node (b). Loops (b) and multiple (c) ingoing arcs in the node are also possible. The later fact is called a reentrance. Note that there is at most one outgoing arc with a certain feature label from every node, but several ingoing arcs with the same feature label are allowed (b). In feature graphs, important is to reach every node from the root node and to label all sink nodes.

It is simple to represent a feature graph as a attribute-value matrix (AVM). AVM represents the nodes of the feature graphs as *structures* in such way that non-sink nodes are enclosed in square brackets, but sink nodes are simply represented with their labels. The information about features (outgoing arcs) and values (destination nodes) are structured like a matrices with two columns, the first column stands for the attribute[6] list and the second one for list of the corresponding feature values. In AVM, each structure is labeled similarly (up to isomorphism) to the names of the corresponding nodes[7] in the feature graph. It is common to use natural numbers for labeling the structures in AVMs. Reentrance and loops are expressed in terms of labels. There are examples of AVMs (in Fig.2.2) corresponding to the feature graphs in Fig.2.1:

[6]We will make no difference between attributes and features while talking about the feature structures.

[7]There is one-to-one mapping between the nodes of the feature graph and structures of the corresponding AVM.

$$\boxed{0}\begin{bmatrix}\textsc{born\_in} & \boxed{1}\ \textit{1990}\\ \textsc{have} & \boxed{2}\begin{bmatrix}\textsc{prod.\_in} & \boxed{4}\ \textit{1986}\\ \textsc{mark} & \boxed{3}\ \textit{Kia}\end{bmatrix}\end{bmatrix} \qquad \boxed{0}\begin{bmatrix}\textsc{feat}_1 & \boxed{0}\\ \textsc{feat}_2 & \boxed{1}\begin{bmatrix}\textsc{feat}_1 & \boxed{0}\end{bmatrix}\end{bmatrix} \qquad \boxed{0}\begin{bmatrix}\textsc{born\_in} & \boxed{3}\ \textit{1986}\\ \textsc{have} & \boxed{2}\begin{bmatrix}\textsc{prod.\_in} & \boxed{3}\end{bmatrix}\\ \textsc{sex} & \boxed{1}\ \textit{female}\end{bmatrix}$$

(a) (b) (c)

Figure 2.2: Examples of AVMs corresponding to the feature graphs

In AVMs, the order of the features for each structure is irrelevant as there is no order between outgoing arcs of a certain node in feature graphs.

From AVMs it is easy to see that it models a set of all objects which have some feature-values and those values are also modeled in terms of feature-values, etc. But the representations in terms of AVMs and feature graphs are depended on some irrelevant specific details of a set of labels (in AVMs) or names of nodes (in feature graphs). For example, in AVMs, the value of each label is irrelevant as the AVMs in Fig.2.3 describes the *equivalent* objects and the AVM (c) in Fig.2.2:

$$\boxed{3}\begin{bmatrix}\textsc{born\_in} & \boxed{0}\ \textit{1986}\\ \textsc{have} & \boxed{1}\begin{bmatrix}\textsc{prod.\_in} & \boxed{0}\end{bmatrix}\\ \textsc{sex} & \boxed{2}\ \textit{female}\end{bmatrix} \qquad \boxed{4}\begin{bmatrix}\textsc{born\_in} & \boxed{1}\ \textit{1986}\\ \textsc{have} & \boxed{2}\begin{bmatrix}\textsc{prod.\_in} & \boxed{1}\end{bmatrix}\\ \textsc{sex} & \boxed{3}\ \textit{female}\end{bmatrix} \qquad \boxed{1}\begin{bmatrix}\textsc{born\_in} & \boxed{7}\ \textit{1986}\\ \textsc{have} & \boxed{2}\begin{bmatrix}\textsc{prod.\_in} & \boxed{7}\end{bmatrix}\\ \textsc{sex} & \boxed{3}\ \textit{female}\end{bmatrix}$$

(a) (b) (c)

Figure 2.3: Example of equivalent AVMs

The reason is that the set(!) of labels of each AVM give the same information about the (equality and inequality) relations between the structures of AVMs. In other words, there are isomorphisms between the set of labels of the AVMs. The isomorphic relation satisfies properties of a equivalence relation, hence, in order to abstract from the irrelevant specific details we can partition the set of all concrete feature structures into equivalence classes. Resulted equivalence classes are called *abstract feature structures* or simply *feature structures* (for more details see (Moshier, 1987), (Moshier, 1988)).

$$\begin{bmatrix}\textsc{born\_in} & \textit{1990}\\ \textsc{have} & \begin{bmatrix}\textsc{prod.\_in} & \textit{1986}\\ \textsc{mark} & \textit{Kia}\end{bmatrix}\end{bmatrix} \qquad \boxed{0}\begin{bmatrix}\textsc{feat}_1 & \boxed{0}\\ \textsc{feat}_2 & \begin{bmatrix}\textsc{feat}_1 & \boxed{0}\end{bmatrix}\end{bmatrix} \qquad \begin{bmatrix}\textsc{born\_in} & \boxed{0}\ \textit{1986}\\ \textsc{have} & \begin{bmatrix}\textsc{prod.\_in} & \boxed{0}\end{bmatrix}\\ \textsc{sex} & \textit{female}\end{bmatrix}$$

(a) (b) (c)

Figure 2.4: Examples of AVMs representing abstract feature structures

In Fig.2.4, we give examples of AVMs representing the feature structures. In AVMs, the structural equalities are explicit with the help of labels[8] and structural inequality are represented implicitly - if two structures do not have the same labels then they are unequal. In the later chapters we will be using feature structures[9] representing in terms of AVMs.

[8]It is assumed to use the least unused natural numbers whenever we need to labels the structures with the same label.

[9]The introduction of the subsumption relation and unification operation (in terms of morphisms) on the (concrete) feature structures are out of the scope of this thesis. Instead, we will introduce them on the typed feature structures.

2.1.2 Type hierarchy and signature

A type hierarchy is introduced in order to type the feature structures. It is natural to think about types and the type hierarchy as concepts and the concept hierarchy, respectively; where the hierarchy is an inheritance hierarchy meaning that the subconcepts (subtypes) will inherit all the properties from their superconcepts (supertypes).

Definition: A *type hierarchy* $\langle \mathcal{T}, \sqsubseteq \rangle$ is a finite[10] *bounded complete* partially ordered (BCPO) set.

The type hierarchy is also referred as the *inheritance hierarchy* or the *sort hierarchy*. $\sqsubseteq$ relation is called a subsumption relation and when $t_1 \sqsubseteq t_2$ we say that t_2 subsumes (is greater than or is supertype of) t_1, or t_1 is subsumed by (less than or subtype of) t_2. *Bound complete* property of a partial order means that any low-bounded (aka *consistent*) subset $S \subseteq \mathcal{T}$, i.e., there exists an element $t \in \mathcal{T}$ such that $t \sqsubseteq s$ for every $s \in S$, has the (unique) greatest lower bound (glb)[11] denoted as $\sqcap S$. The existence of the *maximum* (aka *top*) element in $\langle \mathcal{T}, \sqsubseteq \rangle$ is implied from BCPO property: $\emptyset \subseteq \mathcal{T}$ is consistent as any element of $\mathcal{T}$ is less than any element of $\emptyset$, therefore there exists $\top = \sqcap\emptyset$, which is the greatest element, i.e., the maximum element, in $\mathcal{T}$. Also note that for a $S \subseteq \mathcal{T}$ the least upper bound (lub) is defined in the following way:

$$\sqcup S = \sqcap\{t \mid s \sqsubseteq t \text{ for every} s \in S\}$$

only if the glb, i.e., the right part of the equality, exists. In Fig.2.5, there are examples of the valid and invalid type hierarchies. If *sqcap*$\mathcal{T}$ is defined than it represents the minimum element and is denoted as $\bot$. For $S \subseteq \mathcal{T}$, if $\sqcap S = \bot$, than S is called *inconsistent*. In the thesis, we assume that $\bot$
notin$\mathcal{T}$.

$$\begin{tikzcd}
 & \top \arrow[dl] \arrow[dr] & \\
a \arrow[d] \arrow[drr] & & b \arrow[d] \arrow[dll] \\
c & & d
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \top \arrow[dl] \arrow[dr] & \\
a \arrow[dr] & & b \arrow[dl] \\
 & e \arrow[dl] \arrow[dr] & \\
c & & d
\end{tikzcd}$$

(a) (b)

Figure 2.5: Examples of the invalid (a) and valid (b) type hierarchies

The type hierarchy (a) is invalid because $S = \{a, b\}$ is consistent as there exists an element (e.g. c) which is less than each element in S, but $\sqcap S$ is not defined as c and d are not in subsumption relation. The type hierarchy (b) is valid and represents the "fixed" version of (a). In (b), $\sqcap S$ is defined for every consistent subset $S \subseteq \mathcal{T}$.

[10]Sometimes this restriction is not required, but as we are going to talk about its implementation, we assume that the set if finite.

[11]In other words, the type hierarchy is a finite meet semilattice, where the meet of two elements is glb.

In order to specify the set of appropriate features for each type in the hierarchy, the *appropriateness specification* is introduced. We assume that $\mathcal{F}$ is a set of features and $\mathcal{F} \cap \mathcal{T} = \emptyset$. The expression $f(x, y) \downarrow$ means that $f(x, y)$ is defined.

DEFINITION: The *appropriateness specification* on the type hierarchy $\langle \mathcal{T}, \sqsubseteq \rangle$ and the set of features $\mathcal{F}$ is a partial function $\mathcal{A}pprop : \mathcal{F} \times \mathcal{T} \to \mathcal{T}$ such that satisfies two properties:

a) Maximal introduction of features:
For every $f \in \mathcal{F}$, $T_f = \{t \in \mathcal{T} \mid \mathcal{A}pprop(f, t) \downarrow\} \neq \emptyset$, where $\sqcup \mathcal{T}$ is defined and $\sqcup \mathcal{T} \in T_f$;

b) Downward closure / Right monotonicity:
If $t_2 \sqsubseteq t_1$ and $\mathcal{A}pprop(f, t_1) \downarrow$ for some $f \in \mathcal{F}$; then $\mathcal{A}pprop(f, t_2) \downarrow$ and $\mathcal{A}pprop(f, t_2) \sqsubseteq \mathcal{A}pprop(f, t_1)$.

Note that these properties require that (a) every feature is introduced by some unique most general type, (b) the feature is appropriate to all subtypes of this most general type and the type of the feature at the subtype is at most as general as it is for the same feature at supertypes. If $\mathcal{A}pprop(f, t_1) = t_2$, we say that the feature f is appropriate to the type t_1 and the type t_2 is appropriate to the feature f at the type t_1.

DEFINITION: The appropriateness specification contains a *loop* if there exists a sequence $(t_i)_{i=1}^{n+1}$ of types and a sequence $(f_i)_{i=1}^{n}$ of features such that for every i, $1 \leq i \leq n$, $\mathcal{A}pprop(f_i, t_i) = t_{i+1}$, where $t_{n+1} = t_1$.

The type t_1 is called a *cyclictype* and features $(f_i)_{i=1}^{n}$ a cyclic features [Penn, 2000]. The types with cyclic appropriate features do not have finite totally well-typed most general satisfiers, and (Carpenter, 1992) excludes them by prohibiting *appropriateness loops*. As our aim is to give the foundation of the HPSG formalism, which is based on the totally well-typed feature structures, we will also exclude the appropriateness loops in the general definition of the signature.

DEFINITION: The signature is tuple $\langle \mathcal{T}, \sqsubseteq, \mathcal{F}, \mathcal{A}pprop \rangle$, where $\langle \mathcal{T}, \sqsubseteq, \rangle$ is a type hierarchy, $\mathcal{F}$ is a set of features, $\mathcal{A}pprop$ is an appropriateness specification function on $\langle \mathcal{T}, \sqsubseteq, \rangle$ and $\mathcal{F}$ and has no appropriateness loops[12].

There are examples of invalid signatures in Fig.2.6. In the type hierarchy appropriateness specification is expressed by the labels attached to the types in such way, that the fact $\mathcal{A}pprop(\text{F}, t_1) = t_2$ corresponds to attaching $\text{F} : t_2$ label to the type t_1, where the feature names are in uppercase. The feature in bold face denotes its maximal introduction. In the signature (a) (Fig.2.6) H feature has no maximum introduction. In Fig.2.6, (b) represents one of the solutions of (a), but now the downward closure property is violated at type f. In Fig.2.7, (a) meets all requirements to be valid signature except having no appropriateness loops. The sequences (a, f, a) and (G, H) form the loop in (a). The signature becomes valid after adding the extra type k, see (b).

More advanced information about the signature and its influence on theoretical and practical sides of the operations on typed feature-structure are available in (Carpenter, 1992) and (Penn, 2000).

[12]Sometimes loops are not forbidden in the signature but here we forbid as they are not allowed in the implemented signature.

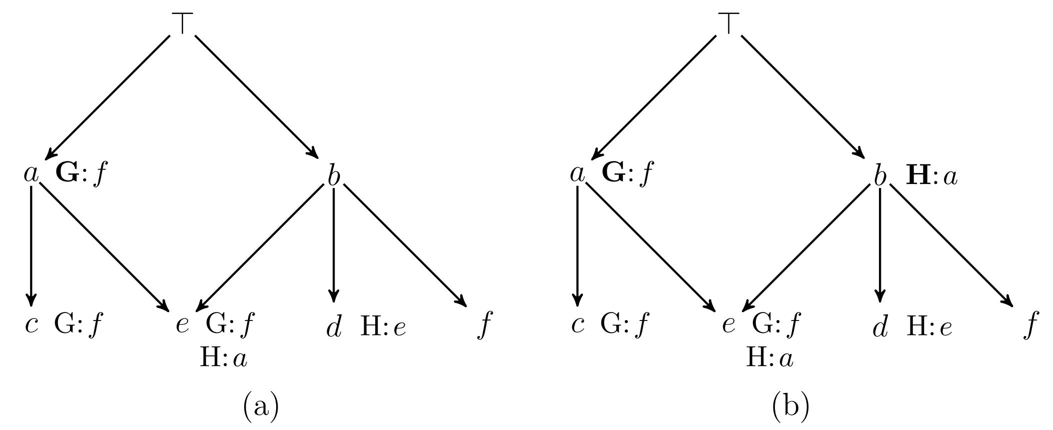


Figure 2.6: Gradually fixing the invalid signature: (a) violates the maximum introduction of features and downward closure, while (b) violates only the later.

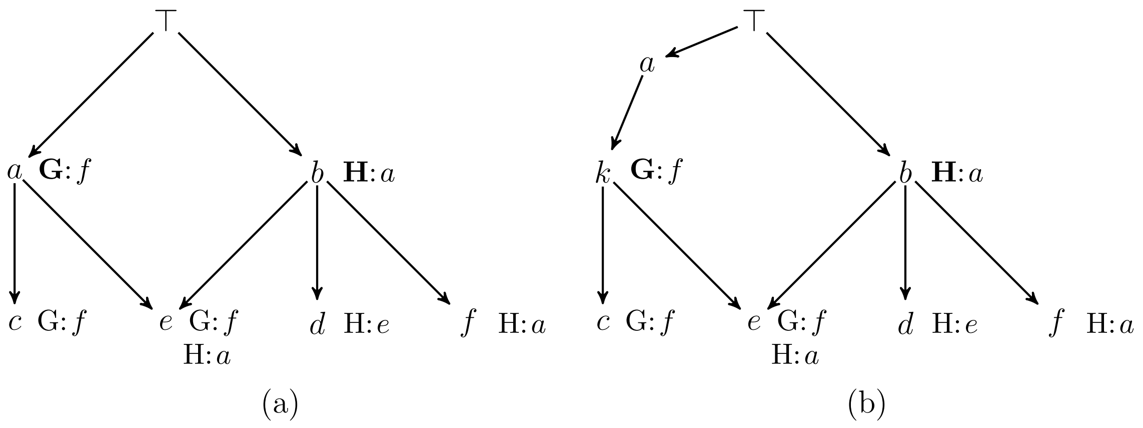


Figure 2.7: Elimination of the loop from the signature: (a) includes the loop, but after adding the new type the loop is eliminated in (b).

2.1.3 Typed feature structures

We introduced feature structures, the type hierarchy $\langle \mathcal{T}, \sqsubseteq \rangle$ and the signature $\langle \mathcal{T}, \sqsubseteq, \mathcal{F}, \mathcal{A}pprop \rangle$. We fix an infinite set of nodes $\mathcal{Q}$ disjoint from $\mathcal{T}$ and $\mathcal{F}$, each element of which is typed by the typing function $\theta : \mathcal{Q} \to \mathcal{T}$. Also, for every $t \in \mathcal{T}$, $\{q \in \mathcal{Q} \mid \theta(q) = t\}$ is infinite. Now we introduce typed feature structures, which the HPSG formalism is based on. A typed feature structure is defined formally as a typed feature graph.

DEFINITION: A *typed-feature structure* (TFS) over $\langle \mathcal{T}, \sqsubseteq \rangle$ and $\mathcal{F}$ is a tuple $\langle Q, q_0, \delta \rangle$:

$Q \subset \mathcal{Q}$ is a finite set;

$q_0 \in Q$ is the root node;

$\delta : Q \times \mathcal{F} \to Q$ is a partial feature value function such that for any $q' \in Q$, there exists a sequences of nodes $\langle q_0, \ldots, q_n, q' \rangle$ and a sequence of features $\langle f_1, \ldots, f_n \rangle$ that $\delta(q_i, f_{i+1}) = q_{i+1}, (i = 0, \ldots n)$, where $q_{n+1} = q'$.

The definition simply says that a TFS is a finite, rooted, directed, connected, labeled graph. The nodes and arcs are labeled with types from $\mathcal{T}$ and features from $\mathcal{F}$, respectively. q_0 is a root node and every node of the graph is reachable from it. The type of the TFS is the type of the root node $t = \theta q_0$. We fix the set of all TFSs and denote it by $\mathcal{TFS}$. The examples of the TFSs (viewed as feature graphs) are given in Fig.2.8, where (a) and (c) are of type *human*, and (b) is of type *a*.

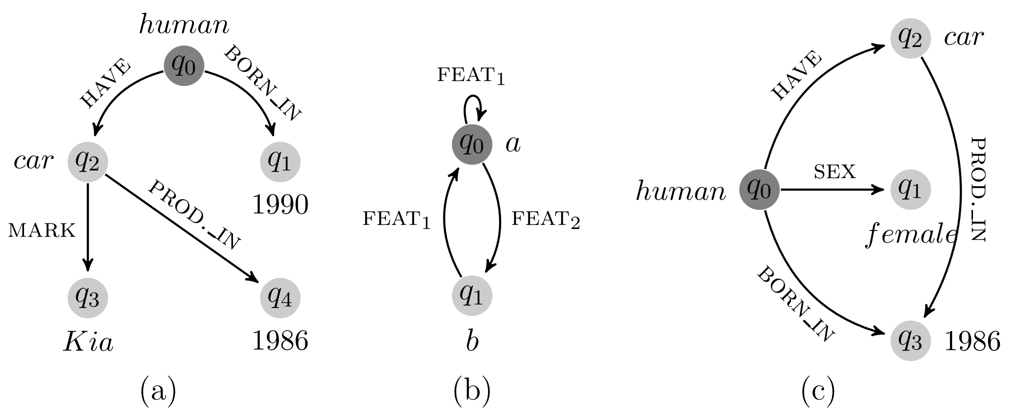


Figure 2.8: Examples of typed feature graphs on $\langle \mathcal{T}, \sqsubseteq \rangle$, where $\mathcal{T} = \{1990, 1986, a, b, car, female, human, Kia\}$, $\sqsubseteq$ is equality relation and $\mathcal{F} = \{\text{FEAT}_1, \text{FEAT}_2, \text{BORN\_IN}, \text{HAVE}, \text{MARK}, \text{PROD.\_IN}, \text{SEX}\}$

The AVM representations of the same TFSs are given in Fig.2.9. Comparing to the untyped AVMs, the types of non-sink nodes are placed in the upper left corner and the types of sink ones substitute for atoms. Since now, we will use AVMs representations for drawing TFSs.

$$\boxed{0}\begin{bmatrix} \textit{human} & \\ \text{BORN\_IN} & \boxed{1}\ \textit{1990} \\ \text{HAVE} & \boxed{2}\begin{bmatrix} \textit{car} & \\ \text{PROD.\_IN} & \boxed{4}\ \textit{1986} \\ \text{MARK} & \boxed{3}\ \textit{Kia} \end{bmatrix} \end{bmatrix} \qquad \boxed{0}\begin{bmatrix} \textit{a} & \\ \text{FEAT}_1 & \boxed{0} \\ \text{FEAT}_2 & \boxed{1}\begin{bmatrix} \textit{b} & \\ \text{FEAT}_1 & \boxed{0} \end{bmatrix} \end{bmatrix} \qquad \boxed{0}\begin{bmatrix} \textit{human} & \\ \text{BORN\_IN} & \boxed{3}\ \textit{1986} \\ \text{HAVE} & \boxed{2}\begin{bmatrix} \textit{car} & \\ \text{PROD.\_IN} & \boxed{3} \end{bmatrix} \\ \text{SEX} & \boxed{1}\ \textit{female} \end{bmatrix}$$

(a) (b) (c)

Figure 2.9: Examples of typed AVMs

DEFINITION: A TFS $\langle Q, q_0, \delta \rangle$ with respect to the signature $\langle \mathcal{T}, \sqsubseteq, \mathcal{F}, \mathcal{A}pprop \rangle$ is *well-typed* iff $\delta(q, f) \downarrow$ implies $\mathcal{A}pprop(f, \theta(q)) \downarrow$, and $\delta(q, f) \sqsubseteq \mathcal{A}pprop(f, \theta(q))$.

i.e., well-typed TFS do not "contradict" the signature: for any arc, the feature (label) of the arc is appropriate to the type (label) of the source node and the appropriate type for the feature at the type of the source node subsumes the type of the target node. The examples of well-typed TFS are given in Fig.2.11 with respect to *signature*1 (Fig.2.10).

DEFINITION: A TFS $\langle Q, q_0, \delta \rangle$ with respect to the signature $\langle \mathcal{T}, \sqsubseteq, \mathcal{F}, \mathcal{A}pprop \rangle$ is *totally well-typed* iff it is well-typed and $\mathcal{A}pprop(f, \theta(q)) \downarrow$ implies $\delta(q, f) \downarrow$.

i.e., totally well-typed TFS "completely follows" the signature: for each type, all appropriate features are labeling the outgoing arcs of each node labeled with this type.
The examples of totally well-typed TFSs are given in Fig.2.11. (a) and (b) are not totally well-typed as the types f and e lack the appropriate features H and

$$\begin{tikzcd}
 & & \top \arrow[dl] \arrow[dr] & & \\
 & a\ \mathbf{G}{:}f \arrow[dl] \arrow[dr] & & b\ \mathbf{H}{:}\top \arrow[dl] \arrow[d] \arrow[dr] & \\
c\ \text{G}{:}f & & e\ \begin{array}{l}\text{G}{:}f\\ \text{H}{:}\top\end{array} & d\ \text{H}{:}e & f\ \text{H}{:}\top
\end{tikzcd}$$

Figure 2.10: The signature: $signature1$

$$\text{(a)}\quad \boxed{0}\begin{bmatrix} e \\ \text{G}\ \boxed{1}\ f \\ \text{H}\ \boxed{2}\ \top \end{bmatrix}
\qquad
\text{(b)}\quad \boxed{0}\begin{bmatrix} a \\ \text{G}\ \boxed{1}\begin{bmatrix} f \\ \text{H}\ \boxed{2}\begin{bmatrix} e \\ \text{H}\ \boxed{3}\ a \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\text{(c)}\quad \boxed{0}\begin{bmatrix} e \\ \text{G}\ \boxed{1}\begin{bmatrix} f \\ \text{H}\ \boxed{2} \end{bmatrix} \\ \text{H}\ \boxed{2}\ \top \end{bmatrix}
\qquad
\text{(d)}\quad \boxed{0}\begin{bmatrix} k \\ \text{G}\ \boxed{1}\begin{bmatrix} f \\ \text{H}\ \boxed{2}\begin{bmatrix} e \\ \text{G}\ \boxed{4}\begin{bmatrix} f \\ \text{H}\ \boxed{3} \end{bmatrix} \\ \text{H}\ \boxed{3}\ \top \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.11: (Totally)well-typed AVMs wrt *signature*1; all (a)-(d) are well-typed, while only (c),(d) are totally well-typed.

G, respectively. (c) and (d) represent totally well-typed versions of (a) and (b), respectively.

2.1.4 Subsumption order and unification operation

The partial order on the type hierarchy induces the partial subsumption order on the set of TFSs. The subsumption order is introduced in terms of morphisms (structure-preserving maps) on TFSs.

DEFINITION: a TFS $F_1 = \langle Q_1, q_1, \delta_1 \rangle$ subsumes a TFS $F_2 = \langle Q_2, q_2, \delta_2 \rangle$ (denoted as $F_2 \sqsubseteq F_1$) if there exists as total function $h : Q_1 \to Q_2$ such that:
$h(q_1) = h(q_2)$;
for every $q \in Q_1$, $\theta(h(q)) \sqsubseteq h(q)$;
for every $q \in Q_1$ and every f feature, if $\delta_1(q, f) \downarrow$, then $\delta_2(h(q), f) \downarrow$ and $h(\delta_1(q, f)) = \delta_2(h(q), f)$.

The h function is called *subsumption morphism* and it "embeds" F_1 into F_2 preserving the structure of F_1: h maps a source node from F_1 to the target node of F_2 which has at least as specific type as the source node has; and if there is a labeled f arc from q to q' in F_1, then there is the f labeled arc from $h(q)$ to $h(q')$ in F_2. In Fig.2.11, (c)$\sqsubseteq$(a) and (d)$\sqsubseteq$(b) the sumsumption morphism in both case is the identity function $h(n) = n$. Note, that there is no subsumption relation between (c) and (d), though (c) represents the *substructure* of (d). The reason is that the root nodes have different types. As you see the subsumption relation does not employ appropriateness specification.

DEFINITION: TFSs F_1 and F_2 are *alphabetic variants* (denoted as $F_1 \sim F_2$) iff

$F_1 \sqsubseteq F_2$ and $F_2 \sqsubseteq F_1$.

The *alphabetic variation relation* is not necessarily equality relation. Two TFSs can be alphabetic variants[13], but different; e.g., (a)$\sim$(b) but (a)$\neq$(b) (in Fig.2.12). The reason is that they have different nodes, though they are typed in the same way and have the same structure. The alphabetic variation relation is an equivalence relation over the TFSs. In order to abstract from the identities of nodes in the TFSs and keep the information only about types of nodes, feature labels of arcs and their structure, the *abstract* TFSs are introduced. An abstract TFS simply represents the equivalence class $[tfs]_\sim = \{t \in \mathcal{TFS} \mid t \sim tfs\}$, therefore the set of all abstract TFSs is a *quotient set* of $\mathcal{TFS}$ *modulo* $\sim$, denoted by $\mathcal{ATFS} = \mathcal{TFS}/_\sim = \{[tfs]_\sim \mid tfs \in \mathcal{TFS}\}$. In Fig.2.12, (a)-(c) are elements of the same equivalence class represented by the (d) abstract AVM (i.e., an AVM of an abstract TFS), which is not any more depended on identities of nodes[14].

$$\underset{\text{(a)}}{\boxed{0}\begin{bmatrix} e \\ \text{G}\ \boxed{1}\begin{bmatrix} f \\ \text{H}\ \boxed{2} \end{bmatrix} \\ \text{H}\ \boxed{2}\ a \end{bmatrix}} \quad \underset{\text{(b)}}{\boxed{1}\begin{bmatrix} e \\ \text{G}\ \boxed{2}\begin{bmatrix} f \\ \text{H}\ \boxed{0} \end{bmatrix} \\ \text{H}\ \boxed{0}\ a \end{bmatrix}} \quad \underset{\text{(c)}}{\boxed{0}\begin{bmatrix} e \\ \text{G}\ \boxed{2}\begin{bmatrix} f \\ \text{H}\ \boxed{3} \end{bmatrix} \\ \text{H}\ \boxed{3}\ a \end{bmatrix}} \quad \underset{\text{(d)}}{\begin{bmatrix} e \\ \text{G}\ \begin{bmatrix} f \\ \text{H}\ \boxed{0} \end{bmatrix} \\ \text{H}\ \boxed{0}\ a \end{bmatrix}}$$

Figure 2.12: Examples of alphabetic variant AVMs: (a)-(c) are alphabetic variants and (d) is an AVM of the abstract TFS representing the equivalence class of (a)-(c).

Note that, the notions of well-typedness and totally well-typedness, and the subsumption relation are correctly transfered from $\mathcal{TFS}$ to $\mathcal{ATFS}$ (for formal proves see (Carpenter, 1992, chap.3). Namely, for every $atfs_i \in \mathcal{ATFS}$ and $tfs_i \in \mathcal{TFS}$, the following facts hold[15]:

$atfs_1$ is (totally) well-typed iff every $tfs_1 \in atfs_1$ is (totally) well-typed;
$atfs_1 \sqsubseteq atfs_2$ iff for every $tfs_1 \in atfs_1$ and $tfs_2 \in atfs_2$, $tfs_1 \sqsubseteq tfs_2$.
$atfs_1 \sqsubseteq atfs_2$ and $atfs_2 \sqsubseteq atfs_1$, iff $atfs_1 = atfs_2$

Finally, now we are at the point to define one of the most important operation on abstract TFSs, namely *unification operation*. After the subsumption relation is defined on $\mathcal{ATFS}$, it make sense to talk about glb of bounded set of abstract TFSs in $\langle \mathcal{ATFS}, \sqsubseteq \rangle$.

Definition: if $A \subseteq \mathcal{ATFS}$ is lower bounded in $\langle \mathcal{ATFS}, \sqsubseteq \rangle$, then the *unification*[16] of A, denoted by $\sqcap A$, is defined $\sqcap A \in \mathcal{ATFS}$, and is the glb of A[17].

Definition: for $A \subseteq \mathcal{ATFS}$, if $\sqcap A$ is defined, we day that A is *consistent*, otherwise it is *inconsistent*.

[13]The alphabetic variation, in other words, is the isomorphism between TFSs

[14]It is assumed to use the least unused natural numbers whenever we need to labels the reentrant structures with the same label.

[15]For simplicity, we keep the same symbols for the subsumption relation on the abstract TFSs.

[16]We use *unification* to refer to the operation or its result.

[17]In some literatures, $\sqcup$ is used for the unification operator. Here, we support $specific \sqsubseteq general$ subsumption relation, hence the unification (resulting the specific values) is denoted as $\sqcap$

The idea of the unification operation is to combine information from the TF-Ss[18], while the subsumption relation compares the informations of two TFSs. Unification is used for enriching the already existing knowledge with some new knowledge via the efficient procedure.

We gave formal definition of the unification operation but it does not say anything how to compute the result of the operation. The unification is recursive procedure and to unify (find glb of) two abstract TFSs, we first unify their types if they are consistent, otherwise the unification fails. Then we unify (here is recursion!) the values of features occurring in both TFSs, otherwise we simply copy a feature with its value. Note also, that we have to take care of the reentances (structure sharing) during the unification. The examples of the unifications of well-typed abstract TFSs wrt *signature*1 (Fig.2.10) are given in Fig.2.13:

$$\left[\begin{array}{ll} a & \\ \text{G} & \left[\begin{array}{ll} f & \\ \text{H} & \left[\begin{array}{ll} a & \\ \text{G} & f \end{array}\right] \end{array}\right] \end{array}\right] \sqcap \left[\begin{array}{ll} e & \\ \text{G} & \left[\begin{array}{ll} f & \\ \text{H} & e \end{array}\right] \\ \text{H} & e \end{array}\right] = \left[\begin{array}{ll} e & \\ \text{G} & \left[\begin{array}{ll} f & \\ \text{H} & \left[\begin{array}{ll} e & \\ \text{G} & f \end{array}\right] \end{array}\right] \\ \text{H} & e \end{array}\right]; \quad \left[\begin{array}{ll} a & \\ \text{G} & \left[\begin{array}{ll} f & \\ \text{H} & \left[\begin{array}{ll} a & \\ \text{G} & f \end{array}\right] \end{array}\right] \end{array}\right] \sqcap \left[\begin{array}{ll} e & \\ \text{G} & \left[\begin{array}{ll} f & \\ \text{H} & d \end{array}\right] \\ \text{H} & e \end{array}\right] = \perp;$$

$$\left[\begin{array}{ll} a & \\ \text{G} & \boxed{0} \left[\begin{array}{ll} f & \\ \text{H} & \left[\begin{array}{ll} a & \\ \text{G} & \boxed{0} \end{array}\right] \end{array}\right] \end{array}\right] \sqcap \boxed{0} \left[\begin{array}{ll} e & \\ \text{G} & \boxed{1} \left[\begin{array}{ll} f & \\ \text{H} & \boxed{0} \end{array}\right] \\ \text{H} & \boxed{1} \end{array}\right] = \boxed{0} \left[\begin{array}{ll} e & \\ \text{G} & \boxed{1} \left[\begin{array}{ll} f & \\ \text{H} & \boxed{0} \end{array}\right] \\ \text{H} & \boxed{1} \end{array}\right]$$

Figure 2.13: Examples of the unification of well-typed abstract TFSs

In Fig.2.13, the second example, the abstract TFSs are inconsistent as the unification fails due to the inconsistence of types a and d. The third example shows how the structure sharing makes the unification "harder" in contrast with the first example. Also, the same example shows the *absorbing* property of the unification which occurs when one operand is subsumed by another.

For more details about the unifications of TFSs can be found in (Carpenter, 1992), where the author discusses the type inference of TFSs and well-typed TFS wrt the signature, the variants of unifications on set of TFSs, well-typed TFS and totally well-typed TFS, and necessary conditions to guarantee that the unification is in the same domain as the operands (closure under the unification).

[18]Note, that the unification operator is also defined for untyped feature structures, but we skip this part here.

2.2 Head-driven phrase structure grammar

First, we list and explain the main characteristics of the grammar formalism. Then we introduce the grammar framework by describing its components – the signature, the lexicon, lexical rules, principles and grammar rules.

2.2.1 Main principles of HPSG

The *Head-driven Phrase Structure Grammar* (HPSG) formalism was introduced by Carl Pollard and Ivan Sag in their work *Information-Based Syntax and Semantics* (Pollard & Sag, 1987). It evolved directly from attempts to modify another grammar formalism – *Generalized Phrase Structure Grammar* (GPSG), but became much more popular than its ancestor formalism. A revised and more complete theoretical framework of HPSG, along with attempts to model a broad range of phenomena in the English grammar, was given in (Pollard & Sag, 1994).

The characteristics of the HPSG formalism you hear first about are that it is a generative but non-derivational grammar formalism, founded on the typed feature structures, that heavily employs the unification operation, has the declarative property, uses a constraint-based and monostratal approach, and is highly lexicalized. Here, we will try to briefly characterize all these properties:

- HPSG (like other *generative* grammar formalisms), is based on a set of rules, attempts to distinguish word combinations constituting grammatical sentences from those constituting ungrammatical ones.
- It is *non-derivational*, as it employs no destructive transformations (unlike transformational grammars), it does not derive sequentially one structure (or representation) from another, but it gradually builds a single large structure.
- The formalism is founded on *the logic of typed feature structures*. We can talk about linguistic objects in terms of descriptions (i.e., formulas) of the logic, interpretations of which are TFSs. This makes HPSG attractive from the computational point of view.
- *The unification operation* on the TFSs (i.e., linguistic objects) serves as a combining operation on the linguistic objects in HPSG. Unifying two linguistic objects results in the larger object containing all the information from the operand objects. In this way, sentences and phrases represent the unification of their lexical constituents.
- HPSG-based grammars are written in a *declarative* way, like a knowledge base of the actual grammar of the language. They license linguistic expressions without detailed instructions about the flow of the licensing process – how to behave in some specific cases.
- The formalism employs *constraints* to forbid ungrammatical expressions. The constraints are expressed in terms of descriptions in the logic of TFSs. If a linguistic expression conforms to all appropriate constraints then it is licensed by the grammar.
- It is considered as *monostratal* as there is only one level for representing the information from different levels of the linguistic structure (phonology, morphology, syntax, semantics and pragmatics), although all these pieces of

information are structured differently in TFSs. Monostratal representation makes the interaction between these linguistic levels easy.

- HPSG is highly *lexicalized* – lexical entities are represented by information-rich structures having all the information about different levels of linguistic structure organized in TFSs, here the information and its geometrical structure is determined by the empirical considerations.

All the above-mentioned and other properties made the HPSG formalism one of the most influential grammar formalism among computational linguists: "there are more people working on and with implemented head-driven phrase structure grammars than with any other linguistic grammar model" (Uszkoreit, 1996).

Moreover, several notable approaches and theories to the language-universals are developed in the framework of HPSG (Uszkoreit, 1996): head-driven approach to valency and phrase structure; binding theory based on obliqueness; different theories on long-distance dependencies; theory of complementation including a theory of raising; theory on agreement; approach to semantics, etc.

Also, some language-specific analyses, that are less naturally described in other grammar formalisms, are implemented successfully in HPSG-based grammars (Uszkoreit, 1996), e.g. parasitic gaps, raising in English, French clitics, German phrase structures including fronting, etc.

There is a substantial volume of research done on the HPSG framework. (Levine & Meurers, 2006) provides an aerial view of the linguistic approach of HPSG, summarizes its formal foundations, and characterizes computational work developed based on the HPSG paradigm; (Uszkoreit, 1996) and [HPSG page] briefly lists the main contributions and ideas of HPSG; short explanations of wide range of characteristics of the formalism are given by the co-founder of HPSG in (Pollard, 1997).

2.2.2 The HPSG framework

Here we introduce the HPSG framework. The modularity of its architecture makes it possible to describe each component of the framework separately: signature, describing the structure of TFSs allowed in the formalism and used in the grammar; lexicon, including basic lexical entries; lexical rules, generating derived lexical entries from basic ones; principles applying to words and phrases, having the ability to express dominance relations; grammar rules similar to those of context-free grammars.

Signature

As we already mentioned, the HPSG formalism is founded on the logic of typed feature structures. Hence, one of the components of the formalism is the signature (i.e., the type hierarchy, the sort hierarchy or the system of signs (Pollard & Sag, 1994), where all these names unambiguously refer to a signature in the confines of HPSG). The HPSG signature exactly represents the signature we have formally introduced in the previous chapter. So, the appropriateness loops are not allowed in the HPSG signature too[19].

[19]From the computational point of view, the loops prevent the termination of the typing process of the feature structures

The signature uses Ferdinand de Saussure's notion of the linguistic sign, where *sign* represents the most general type containing information mostly about phonology, syntax, semantics and sometimes pragmatics. *word* and *phrase* types are two immediate subtypes of *sign* and they inherit features from their supertype. The Fig.2.14 demonstrates a toy HPSG signature,[20] where *sign* has information only about phonology (PHON) and grammatical category (CAT),[21] which are inherited by its subtypes:

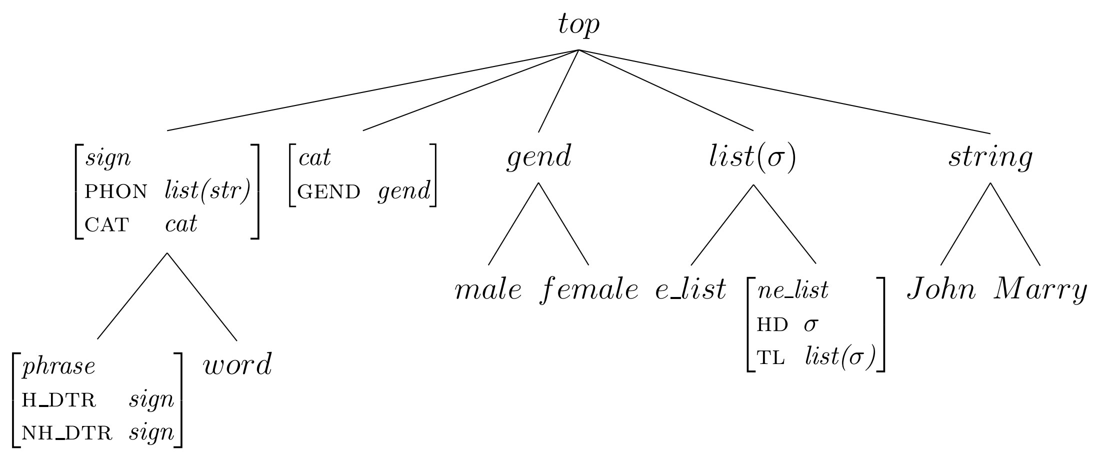


Figure 2.14: A toy HPSG signature

The PHON feature stores the string representation of words or phrases, that is why its type is *list* of *string*s. The *list* type (along with the *set* type) is one of the *recursive types* of the signature, which is claimed to be needed for linguistic descriptions. *list* has two subtypes: *e_list* (an empty list) and *ne_list* (a non-empty list), which itself has two features: HD (the head – the first element of the list) and TL (the tail - a list without the head element). The grammar writer can define a list of some specific types or the most general list type *list*(*bot*).

Two features – H_DTR (the head daughter constituent) and NH_DTR (the non-head daughter constituent) of type *sign* are appropriate for the *phrase* type. These features encode the information about the immediate dominance (ID) relation (a linguistic object of type *phrase* immediately dominates the values of type *sign* of its "daughter" features) and the syntactic dependency relation between the constituents of the phrase (the value of H_DTR depends on the value of H_DTR). The information about daughter constituents can be organized in different ways within structures.

There are also other common language-universal and language specific types in the HPSG signature. We will deal with those types in the later chapters. Meanwhile, note that the hierarchical organization of linguistic information plays a significant role in predicting the impossibility of certain kinds of linguistic phenomena, thus makes grammar writing easier. It is also worthwhile to emphasize that grammar writers are free to introduce new types and features in the signature, modify the type hierarchy or the appropriateness function, or discard some

[20]The signatures of powerful grammars are much more rich with types and features associated with each type.

[21]It is common to use the same name for the feature and its type.

types and features to get the valid signature in the end.[22]

Lexicon

The lexicon in the HPSG framework is a set of lexical entities representing descriptions of the logic of the typed feature structures. As we already mentioned in the principles of HPSG, lexical entries encode rich lexical information in their structure. The fashion of information organization in the structures can be different in different HPSG-based grammars.

$$
\begin{bmatrix}
\textit{word} & \\
\textsc{phon} & \textit{loves} \\
\textsc{synsem} & \begin{bmatrix}
\textit{synsem} & \\
\textsc{cat} & \begin{bmatrix}
\textit{cat} & \\
\textsc{head} & \textit{verb} \\
\textsc{val} & \begin{bmatrix}
\textit{val} & \\
\textsc{subj} & \langle \boxed{1}^{\,nom} \rangle \\
\textsc{comps} & \langle \boxed{2}^{\,acc} \rangle
\end{bmatrix} \\
\textsc{gap} & \langle\, \rangle
\end{bmatrix} \\
\textsc{arg-st} & \langle \boxed{1}\, NP_i^{3sing}, \boxed{2}\, NP_j \rangle \\
\textsc{sem} & \begin{bmatrix}
\textit{sem} & \\
\textsc{mode} & \textit{prop} \\
\textsc{index} & s \\
\textsc{restr} & \left\langle \begin{bmatrix}
\textit{pred-love} & \\
\textsc{sit} & s \\
\textsc{lover} & i \\
\textsc{loved} & j
\end{bmatrix} \right\rangle
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
;\ \mathrm{NP}_i^{agr,case} =
\begin{bmatrix}
\textit{synsem} & \\
\textsc{cat} & \begin{bmatrix}
\textit{cat} & \\
\textsc{head} & \begin{bmatrix}
\textit{noun} & \\
\textsc{case} & \textit{case} \\
\textsc{agr} & \begin{bmatrix}
\textit{agr} & \\
\textsc{pers} & \textit{pers} \\
\textsc{num} & \textit{num}
\end{bmatrix}
\end{bmatrix} \\
\textsc{val} & \begin{bmatrix}
\textit{val} & \\
\textsc{subj} & \langle\, \rangle \\
\textsc{comps} & \langle\, \rangle
\end{bmatrix}
\end{bmatrix} \\
\textsc{sem} & \begin{bmatrix}
\textit{sem} & \\
\textsc{mode} & \textit{ref} \\
\textsc{index} & i \\
\textsc{restr} & \textit{list(pred)}
\end{bmatrix}
\end{bmatrix}
$$

Figure 2.15: An AVM corresponding to the lexicon entry *loves*, where *NP* represents the shortcut for the AVM corresponding to the noun phrase

For the demonstration we give an AVM,[23] corresponding to the lexical entity *loves* (Fig.2.15). The way of organizing the information in the AVM is based on the English grammar developed in (Sag *et al.*, 2003). The AVM includes information about phonology, syntax (CAT) and semantics (SEM). The syntax is expressed by the *cat* structure, containing information about the part of speech (POS), syntactic valency and syntactic gaps, where ⟨ and ⟩ are delimiters of the type *list*. The POS information is a value of the feature HEAD, as the value of HEAD is percolated along the head projection (due to *Head Feature Principle*), the POS information is also available on the phrase level. Structure sharing between the semantic section (the value of SEM), the argument structure (the value of ARG-ST) and valency *val* can be viewed as a syntax-semantic interface, which is easily achieved in the monostratal grammar framework. Note that the type of values of ARG-ST, SUBJ, COMPS and GAP is *list*(*synsem*). The list value of ARG-ST is a concatenation of the list values of SUBJ, COMPS and GAP features.[24] *NP* represents a shorthand of the AVM of the noun phrase, which itself contains the information about the grammatical case, person and number categories. In the grammar of (Sag *et al.*, 2003), the situation semantics are used to encode the semantic information. Therefore, the semantics of *loves* is a propositional

[22]Though some implementation platforms may insist on the existence of some universal types like *list* and *top*

[23]A slightly simplified version than in (Sag *et al.*, 2003).

[24]The feature is used to cover the gaps phenomena in the grammar, for details see (Sag *et al.*, 2003).

situation identified with the index. The situation contains the information about the situation, in which the referent i is in the love relation with the referent j. Note that on the semantic level, the noun phrase is expressed as a reference situation having the index and the predicate restriction on this index. More detailed explanation of each feature and type can be seen in (Sag *et al.*, 2003).

$$word \rightarrow Lexical\text{–}Entry_1 \vee Lexical\text{–}Entry_2 \vee \ldots \vee Lexical\text{–}Entry_n$$

Figure 2.16: The Word Principle

The basic HPSG lexicon can be defined by the *Word Principle* given in Fig.2.16, where each *Lexical–Entry* is a description of the kind we saw in Fig.2.15 (Levine & Meurers, 2006).

Deciding how to organize the information in the lexical entity is up to grammar writer, as it depends on the structure of the signature.

Lexical rules

It would be quite redundant if we had to declare all lexical entities in the lexicon, like *love*, *loves*, *loving*, *loved*, etc, in the fashion shown in Fig.2.16. Lexical rules represent a mechanism which allow us to add just one lexeme[25] for *love* and the applicable lexical rules will produce all necessary inflected (or also, derived) word forms of the lexeme.

The HPSG architecture supports two kinds of lexical rules and they serve for different goals. The first kind of lexical rules can be viewed as principles having the format of the implication between two descriptions. The lexical rule applies to all TFSs satisfying the antecedent description and they are required to satisfy the implied description of the rule. By satisfying the implied description, the TFSs become more specific, hence resolving some of their general types to more specific types. For example, according to this kind of lexical rule, nominative case is assigned to the subject of all finite verbs in (Pollard & Sag, 1994). The lexical rules of this kind are used to encode the *vertical generalizations* (according to (Flickinger, 1987)) between the words, i.e., to modify similarly some class of lexical entries sharing some general properties. As the rules have a format of principles (see the next section) we call them *lexical principles*.

Lexical rules of another kind are used to express the (asymmetric) relation between sets of lexical elements. Unlike the vertical lexical rules, these rules produce new lexicon entries. They behave like a functions on some specific domain of TFSs which satisfy the input description of the lexical rule. The rule also contains the instructions how to create the new TFSs using the input. So, this kind of rules would generate the inflected or derived word forms from the input (in our case, for *love*). We will refer to such kind of rules as *horizontal lexical rules*, as they are capturing the *horizontal generalizations* between the word forms. The final lexicon, which will contain the initial lexicon and all lexical entries, resulting from the application of the vertical and horizontal lexical rules on the initial lexicon, is called the *Full Lexicon*.

[25]We call this kind of lexical entity lexeme (opposed to lemma, which represent canonical form of the lexeme) as it potentially represent the linguistic lexeme – a set of inflected word forms of the same word

Further formal details about lexical rules in HPSG and their implementation can be found in [Meurers, 1994, 1995, 2000, 2001] and [Meurers & Minnen, 1997]. [Briscoe & Copestake, 1999] gives a different view on the lexical rules; the authors formulate both types of lexical rules as TFSs and hence there is no need of introducing a new device.

Principles

We have already mentioned lexical principles and their format in the previous section. Another kind of principles of the HPSG framework will be discussed in the section. Both kinds of principles are commonly referred as grammar principles, but in this work we distinguish these two principles and call *grammar principles* those which are beyond lexical principles (i.e., apply not to lexical entries, but to phrases).

Like lexical principles, *grammar principles* also have the form of implication. In general, they apply to a phrase (i.e. TFS of type *phrase*) which satisfies the constraints of the antecedent of the principle and force it to satisfy the consequent of the principle too. If a phrase satisfies the antecedent and fails the consequent of the grammar principle, then the phrase is not licensed by the grammar.

There are several common grammar principles for HPSG-based grammars. One of the most important ones is the *Head Feature Principle* (HFP), which ensures that in a headed phrase (i.e., a phrase having a head daughter), the head feature of the phrase and the head daughter are identical. In other words, HFP percolates the head information along the head projection (Fig.2.17):

$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS} \ \ \textit{headed-structure} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SYNSEM} \mid \text{CAT} \mid \text{HEAD} & \boxed{0} \\ \text{DTRS} \mid \text{HEAD-DTR} \mid \text{SYNSEM} \mid \text{CAT} \mid \text{HEAD} & \boxed{0} \end{bmatrix}$$

Figure 2.17: Head Feature Principle

The HFP principle makes it easy to check the agreement between constituents. Because of the importance of this principle and the head information, the grammar formalism obtained the prefix head-driven in its name.

From other principles, it is worthy to mention the *immediate dominance principle* (IDP), which expresses the immediate dominance relation between nodes of the phrase structure trees; the *Valency Principle* and the *Argument Realization Principle* (ARP).

Grammar rules

The HPSG grammar rules are used to express the Linear Precedence (LP) relation between the nodes of the phrase structure trees. Grammar rules represent the context-free rules, where they rewrite descriptions instead of symbols. On the left-hand side of the rule, there is a description for the mother category and on the other side the sequence of descriptions of daughters. For example, Fig.2.18 shows the the *Head-Specifier Rule* (HSR) for English:
In case of the strict word order, the grammar rules are used, otherwise grammar principles are preferable. Note that, unlike context-free grammars, HPSG distinguishes the ID and LP relations and it took this property from the ancestor GPSG formalism.

$$\begin{bmatrix} \textit{phrase} & \\ \text{SYNSEM}|\text{CAT}|\text{VAL}|\text{SPR} & \langle\ \rangle \\ \text{DTRS}|\text{H\_DTR} & \boxed{0} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{SYNSEM} & \boxed{1} \end{bmatrix} \quad \boxed{0}\begin{bmatrix} \text{SYNSEM}|\text{CAT}|\text{VAL} & \begin{bmatrix} \text{SPR} & \langle \boxed{1} \rangle \\ \text{COMPS} & \langle\ \rangle \end{bmatrix} \end{bmatrix}$$

Figure 2.18: Head-Specifier Rule

In the end, we can say that an expression is licensed by HPSG if it satisfies all principles and represents a lexicon entry or is resulted by some grammar rule application.

In short, we can express this fact as a formula:

$$\textit{licensed\_expressions} \rightarrow P_1 \wedge \ldots \wedge P_i \wedge (L_1 \vee \ldots \vee L_j \vee R_1 \vee \ldots \vee R_k)$$

The uniform structure and the formal foundation of the formalism makes it easy to understand how the grammar works. Its expressiveness keeps itself attractive to the computational linguists. There are different approaches developed within the HPSG framework for different languages. Wide coverage HPSG-based grammars are written for the languages: English, German, Japanese, Mandarin Chinese, Maltese, Persian, etc. and several platforms were created for implementing these HPSG-based grammars.

2.3 TRALE – an HPSG-based grammar implementation platform

The purpose of this section is to provide basic information about the TRALE system. We start out with an overview on how to run TRALE and get started, then introduce the syntax of the signature and descriptions, how to write a simple HPSG lexicon, lexical rules, principles and grammar rules.

2.3.1 TRALE system

TRALE is a grammar implementation platform that was developed as a part of the MiLCA project in 2003 (Meurers *et al.*, 2002). It is an extension of the Attribute Logic Engine[26] (ALE) system written in SICStus Prolog 3.8.6. and based on the logic of (Carpenter, 1992). TRALE has all of the functionality of ALE plus some extras (most importantly, complex-antecedent constraints). The main goal of this system is to be as faithful as possible to "hand-written" HPSG, and, yet, to incorporate maximally efficient processing "behind the scenes" (Melnik, 2007).

For the grammar development we use the beta version of 2003, so called "old TRALE"[27]. The recent version of the TRALE ("new TRALE") is released in 2008. There are two major factors made us to choose the old version of TRALE. First, the SICStus Prolog which is required to run "new TRALE" is not available for free and second, the newer version is not documented[28]. On the other hand, there is a standalone version of "old TRALE"[29], which does not require the SICStus Prolog and it is documented. One drawback of the standalone version is that it needs to be restarted after some specific errors arise. The main difference between new and old version is the speed, which is not our main goal in the implantation of the grammar.

The TRALE grammar is physically divided in two parts – the signature and the theory, where the former simply represents the signature of the TFSs and later is remaining part of the grammar. First, we introduce the signature and then the theory of the TRALE grammar.

2.3.2 Signature

TRALE physically separates the signature, where types are listed and appropriateness conditions are stated, from the theory. Under TRALE's approach the signature defines the domain of the linguistic objects and the theory distinguishes between those objects that are admissible in the language, and those that are not (Melnik, 2007).

First, we describe the syntax of the TRALE signature and other specifications, then the subtype covering property, which is characteristic for the HPSG framework. This is done in the way of (Penn & Abdolhosseini, 2003a).

[26] ALE is one of the oldest systems still being used for the implementation of HPSG grammars: http://www.cs.toronto.edu/~gpenn/ale.html

[27] http://milca.sfs.uni-tuebingen.de/A4/Course/trale/

[28] According to developers, the changes are minor, but they can be quite confusing if you are unaware of them, and there is no easy way to find out.

[29] http://hpsg.fu-berlin.de/Software/Trale/

Syntax of the signature

TRALE signature is equivalent to the formal signature (hence, to the HPSG signature too). To recall from the previous sections, signature is a finite BCPO set, with the appropriateness function without loops. There are two ways of declaring the signature in TRALE. One is in the style of ALE and another in the style of TRALE. We will introduce the latter as its is very intuitive and intended to resemble the HPSG signature.

The TRALE signature is declared in the separate text file with the default name `signature`. In Fig.2.19, the TRALE signature corresponds to the one of HPSG from Fig.2.14. It is mandatory to start the signature with `type_hierarchy` and finish with (`.`) on the new line. The most general type have to be declared with the name `bot`[30], hence the TRALE signature supports the top-down organization of the type hierarchy. Each type is defined on a separate line following the feature-value pairs in the style `feature:type`, separated by whitespace. The indentation of subtypes should be consistent in the whole signature, i.e., for any type starting at the column C, all its immediate subtypes should start at the column $C+n$, where n is the constant for the signature ($n = 2$ is a common value). Types and features represent the atoms in the Prolog's sense, meaning to start with a lower-case letter and contain only letters, digits and the underscore character. As you see, the same string can be used as a type and as a feature, TRALE will treat them properly.

```
type_hierarchy
bot
  sign  phon:list  cat:cat
    phrase  h_dtr:sign  nh_dtr:sign
    word
  cat  gend:gend
  gend
    male
    female
  list
    e_list
    ne_list  hd:bot  tl:list
.
```

Figure 2.19: The TRALE signature corresponding to the HPSG signature in Fig.2.14

In the TRALE signature, the appropriateness specification is expressed in terms of attaching the feature to the most general type which the feature is appropriated (the maximal introduction property). The subtypes inherit the features and their type restrictions from their supertypes (the right monotonicity property). A type may never occur more than once as the subtype of the same supertype. It can, however, occur as a subtype of two or more different supertypes, i.e., multiple inheritance, it is advised to prefix the second appearance of the type with (`&`), in order to prevent unintended multiple inheritance (Fig.2.20).

[30]The most general type is called *bottom* in fashion of (Carpenter, 1992).

The signature allows the special kind of types called `a_/1` atoms. These types are implicitly declared by TRALE. They are featureless types and can be viewed as Prolog terms.

```
type_hierarchy
bot
  answer  phon:(a_ _)
    yes
    no
    undef
  bool
    true
    false
    &undef
.
```

Figure 2.20: Use of multiple inheritance and `a_/1` atom in TRALE

`a_/1` atoms are used to prevent the signature from redundant types. For example, we avoid defining *Marry* and *John* subtypes of *string* type (Fig.2.14) in the signature (Fig.2.19). The kind of types and word tokens in general will be defined in the lexicon and they will be automatically interpreted by TRALE as types of `a_Marry`, `a_John` and `a_any_word_token`. `a_ _` atom stands for the most general type subsuming all `a_/1` atoms and directly subsumed by `bot`. `a_/1` atoms are often put in parentheses for better reading (Fig.2.20).

Subtype covering

TRALE assumes that subtypes exhaustively cover their supertypes, i.e., that every object of a non-maximal type, t, is also of one of the maximal types subsumed by t. This is only significant when the appropriateness conditions of t's maximal subtypes on the features appropriate to t do not cover the same products of values as t's appropriateness conditions (Penn & Abdolhosseini, 2003a).

This property of the typing is taken from HPSG, where it is called the *closedworld* assumption. This assumes that every object's type should be explicitly defined in the type hierarchy. For better understanding we give the example from HPSG given in (Penn & Abdolhosseini, 2003a). English verbs are typically assumed to have the following two features in order to distinguish auxiliary verbs from main verbs and also to show whether a subject-auxiliary inversion has taken place Fig.2.21.

$$\begin{bmatrix} \textit{verb} & \\ \textsc{aux} & \textit{bool} \\ \textsc{inv} & \textit{bool} \end{bmatrix} = \left\{ \begin{bmatrix} \textit{verb} & \\ \textsc{aux} & \textit{plus} \\ \textsc{inv} & \textit{plus} \end{bmatrix}; \begin{bmatrix} \textit{verb} & \\ \textsc{aux} & \textit{plus} \\ \textsc{inv} & \textit{minus} \end{bmatrix}; \begin{bmatrix} \textit{verb} & \\ \textsc{aux} & \textit{minus} \\ \textsc{inv} & \textit{minus} \end{bmatrix}; \begin{bmatrix} \textit{verb} & \\ \textsc{aux} & \textit{minus} \\ \textsc{inv} & \textit{plus} \end{bmatrix} \right\}$$

Figure 2.21: Subtype covering for English verbs

The values for the AUX and INV features are taken to be of type bool (a). However, note that there cannot be any verbal type like (e). That is, there are no verbs in English that can occur before the subject and not be auxiliaries. With the help of the subtype-covering property we can prevent (e) with the following type hierarchy (Fig.2.22):

```
type_hierarchy
bot
  bool
    plus
    minus
  verb  aux:bool  inv:bool
  aux_verb  aux:plus  inv:bool
  main_verb  aux:minus  inv:minus
.
```

Figure 2.22: Signature preventing the undesired types via sub-type covering

The support for the subtype covering is one of the distinguishing property of TRALE from ALE. The reader can find more information about the TRALE signature and its types in (Penn & Abdolhosseini, 2003a).

2.3.3 Theory

Theory is the name for the "real" grammar – all components of HPSG, except Signature. TRALE defines Theory in the separate file text file (with default name `theory.pl`[31]). In Theory, it is possible to declare the following components of HPSG: macros for descriptions on TFSs, lexical entries of the lexicon, lexical rules, grammar rules, principles, functional descriptions, definite clauses and few other constructs.

Though Theory is defined in the separate file, TRALE allows us to break up it in several files (e.g. separate file for each component of the grammar,) and, in the style of Prolog, import these files in the main file representing the theory file. So, we will also break the introduction to TRALE Theory into several brief descriptions of the individual components. Here, we will not give formal syntax of TRALE Theory, only give the intuition on the format demonstrated with examples. For further details, we refer readers to the user's guide (Penn & Abdolhosseini, 2003a).

Descriptions, variables and macros

Descriptions of the TFSs have simple intuitive representation in TRALE. The best way to explain their syntax is to give the example. TRALE description of the lexical entry *book* (Fig.2.23) is given in List.2.1:
We break the description into lines and indent its features for better readability so that it resembles the TFS. The given description simply represents the conjunction of the constraints on the TFSs. As in Prolog, (,) is a conjunction operator in TRALE. The type is conjugated with its constrained features and features are delimited by (:) from their type constraints. Descriptions end with (.) in the end.

[31]Theory is written in the Prolog file with the file extension .pl.

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \textit{book} \\ \text{CAT} & \begin{bmatrix} \textit{cat} & \\ \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \textit{case} \\ \text{AGR} & \begin{bmatrix} \textit{agr} & \\ \text{PERS} & \textit{per3} \\ \text{NUM} & \textit{sing} \end{bmatrix} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \textit{val} & \\ \text{SUBJ} & \langle\,\rangle \\ \text{COMPS} & \langle\,\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.23: An AVM for the lexicon entry *book*

Listing 2.1: The description for *book*

```
(word,
    phon:(a_ book),
    synsem:(synsem,
        cat:(cat,
            head:(noun,
                case:case,
                agr:(agr,
                    pers:per3,
                    num:sing)),
            val:(val,
                subj:e_list,
                comps:e_list)))).
```

As in HPSG, all TFSs are totally well-formed, TRALE instantiates all descriptions to the most general totally well-typed TFSs satisfying the description[32]. Hence, some redundant constraints can be skipped in the description without loosing any information. List.2.1 and List.2.2 express equivalent descriptions, but the latter is more economic one. Constraints `synsem`, `cat`, `case:case`, `agr`, `val` were redundant.

Listing 2.2: The economic description for *book*

```
(word,
    phon:(a_ book),
    synsem:cat:head:(noun,
                        agr:(pers:per3,
                             num:sing)),
               val:(subj:e_list,
                    comps:e_list)).
```

TRALE allows the use of variables over the descriptions. They start with an

[32]This is done by the *type inference function*. For more theoretical details about it, see (Carpenter, 1992)

uppercase letter. As you see in List.2.3, variables are available on `a_/1` atom level too, but you need to prefix the variable with `a_`. TRALE instantiates the variables to the most general TFSs satisfying all constraints. Therefore, in List.2.3, variables `S` and `N` will be instantiated to `synsem` and `num` types, respectively. TRALE will instantiate `Y` to `e_list` (opposed to `list`), because the second occurrence of the variable is constrained by `e_list` type in the value of `comps`. TRALE does structure sharing in TFSs by using several occurrences of the same variable, like in the case of variable `Y`. Note that due to the difference between structure sharing and token identity, the value of `val` in our recent example (List.2.3) is not the same as the one in List.2.1 or 2.2.

Listing 2.3: the description including variables and structure sharing

```
(word,
     phon:(a_ X),
     synsem:(S,
          cat:head:(noun,
                         agr:(pers:per3,
                              num:N)),
                val:(subj:Y,
                      comps:(Y, e_list))).
```

Macros are shorthands of the descriptions and are useful not only as shorthands. They are very handy during grammar writing as they save the grammar writer from writing whole descriptions. Note that macros can be used within the definition of other macros. An example of a macro definition is given in List.2.4:

Listing 2.4: Defining a macro for nouns having complements

```
noun_word_comp (Num, Case, Comps-ne_list) :=
(word,
     synsem:cat:head:(noun,
                          case:Case,
                          agr:(pers:per3,
                               num:Num)),
                val:(subj:e_list,
                      comps:Comps)).
```

The macro name is on the left-hand side of (`:=`) and its description is on the right-hand side. In our example, macro `noun_word_comp` abbreviates the description on the right-hand side. Moreover, a description can have variables inside and they will be instantiated to the arguments of the macro. Note that the order of the arguments in the macro does not depend on their positions in the description. When using a macro as a description, it should be prefixed with (`@`). For example, the description expressed by the defined macro (List.2.5)
corresponds to the TFSs for the 3rd person, singular, noun word with non-empty complement list and in any grammatical case. Also, it is possible to restrict the arguments of macros to more specific types than they are inferred from the description (like `Comps` to `ne_list`, otherwise it would be restricted to `list`).

Listing 2.5: Usage of the macro as a description

```
@noun_word_comp (sing, case, ne_list).
```

We stop our introduction here. The reader can find the complete syntax of the descriptions described in Backus Normal Forms (BNF), macro hierarchies, automatic generation of macros and other details about these in (Penn & Abdolhosseini, 2003a).

Lexicon

The structure of the lexicon is very simple. It represents a collection of lexical entries. In TRALE, lexical entries are simply listed in the lexicon file. The example of declaring a lexical entry is given in List.2.6.

Listing 2.6: Defining a lexical entry for *book*

```
book ~~> (word,
              synsem:cat:head:(noun,
                                    agr:(pers:per3,
                                         num:sing)),
                        val:(subj:e_list,
                             comps:e_list)).
```

Using this instruction, TRALE automatically adds a TFS (List.2.1) to the lexicon. There is a shorter way to declare lexical entries using macros. The underlying declaration (List.2.7) would give the same result like the one in List.2.6.

Listing 2.7: Defining a lexical entry for *book* using the macro

```
book ~~> @noun_word(sing, _).
```

where, `noun_word` macro is defined as in List.2.8 and _ is an *anonymous variable* (like in Prolog). In this context, it would have the same result if we gave `case` type as an argument instead of _.

Listing 2.8: The macro for nouns without complements

```
noun_word(Num, Case) :=
(word,
    synsem:cat:head:(noun,
                      case:Case,
                      agr:(pers:per3,
                           num:Num)),
               val:(subj:e_list,
                    comps:e_list)).
```

Lexical rules

TRALE supports lexical rules. The format of the rules are simple and resembles the "hand written" rules, i.e., re-writing rules. The given example (List.2.9) of the pluralization lexical rule for English nouns (a little bit modified version than in (Penn & Abdolhosseini, 2003a, chap.6)) completely demonstrates the format and the capabilities of lexical rules in TRALE.

Listing 2.9: The pluralization lexical rule for English nouns

```
plural_n lex_rule
@word_noun(sing, Case)
**>
@word_noun(pl, Case)
if Case = nom
morphs
goose becomes geese,
[k,e,y] becomes [k,e,y,s],
(X,man) becomes (X,men),
(X,F) becomes (X,F,es) when fricative(F),
(X,ey) becomes (X,[i,e,s]),
X becomes (X,s).
% prolog predicates for fricatives
fricative([s]).
fricative([c,h]).
fricative([s,h]).
fricative([x]).
```

`lex_rule` is a keyword for defining the lexical rules and it follows directly to the name of the lexical rule. `**>` stands for the implication between the input and output descriptions. It is optional to define an extra condition for the lexical rule application. Extra conditions are preceded by `if` keyword. `morphs` keyword indicates the start of the morphing phase of the phonology, where `becomes` indicates the direction of the morphing procedure and `when` puts Prolog conditions on the morphing.

The rule applies to lexical entries, TFSs of which satisfy the input description. After satisfaction of the input description, the condition of the rule is checked and the output description is generated. Note that during these procedures some variables (if there are any) are instantiated (like the variable `Case` to the nominative case type `nom`). In the morphing part, atoms, sequences, lists and variables on atoms can be used. It is important to mention that lists have to contain only single character atoms or variables.

Note that this format of lexical rules is adopted from the ALE system. In TRALE, another format of lexical rules is also available and it is intended for the lexical rule compiler. More details about TRALE lexical rules are available in (Penn & Abdolhosseini, 2003a, chap.6,7). Unfortunately, the part of lexical rules are not documented as well as it should be. (Melnik, 2007) compares TRALE lexical rules to "hand-written" ones and talks about information "carry over" issue.

Principles

TRALE (unlike ALE) provides a natural way of defining principles, i.e., they are defined as implications between two descriptions, and they are called complex antecedent constraints. For completeness, we give an example of HFP (Fig.2.17) in TRALE (List.2.10):

Listing 2.10: Head Feature Principle in TRALE

```
(phrase,
     dtrs:headed-structure)
*>
(synsem:cat:head:X,
 dtrs:head-dtr:synsem:cat:head:X).
```

For more details about complex antecedent constraints, we refer the reader to (Penn *et al.*, 2003) and (Penn & Abdolhosseini, 2003a).

Grammar rules

TRALE uses ALE's grammar rules. The rules have a form of implication, where the antecedent description represents the mother category and the consequent is a list of daughter categories with possible procedural attachment (see (Penn & Abdolhosseini, 2003a, chap.6.4)). The implemented version of HSR (Fig.2.18) is given in List.2.11:

Listing 2.11: Head-Specifier Rule in TRALE

```
head_specifer_rule ##
     (phrase,
           synsem:cat:val:spr:e_list ,
           dtrs:h_dtr:X)
===>
     cat> (synsem:Y),
     cat> (X, synsem:cat:val:(spr: [Y],
                                  comps:e_list)).
```

`cat>` is used for single daughters and `cats>`, for a list of daughters whose length is not determined until run-time. Procedural attachments are preceded by `goal>`. Grammar rules are also involved in the generation process. Further details about the syntax of grammar rules and ALE generator can be read in (Penn & Abdolhosseini, 2003a).

Definite clauses and functional descriptions

TRALE allows the use of ALE's definite clauses and functional descriptions. TRALE's definite clauses are similar to those in Prolog. The only significant difference is that first-order terms are replaced with descriptions of feature structures. List.2.12 demonstrates the definite clauses for the simple list append predicate.

Listing 2.12: The list append predicate in TRALE

```
append(e_list, (L, list), L) if true.
append((hd:H, tl:T1), (L, list), (hd:H, tl:T2)) if append(T1,L,T2).
```

The predicate is true iff the third argument is a list concatenation of the first and the second arguments. The similar relation to the append relation can be encoded in terms of functional descriptions (List.2.13). Functional descriptions simply represent functions mapping description to descriptions.

Listing 2.13: The functional description append

```
fun append(+,+,-).
append(e_list, (L, list), L) if true.
append((hd:H, tl:T1), (L, list), (hd:H, tl:T2)) if append(T1,L,T2).
```

The grammar writer should be careful while defining definite clauses or functional descriptions in the grammar, as they can cause non-terminating processes. For further details, we refer the reader to (Penn & Abdolhosseini, 2003a).

Test suite

A test suite is used to check how completely the grammar covers some language phenomena. A test suite is a sequence of tests. It can be defined in a separate file and imported in Theory. The test has the following format (List.2.14):

Listing 2.14: The format of the TRALE test

```
t(Nr,"Test_Item",Description, Solutions,'Comment').
```

where arguments of `t/5` are: the ID number of the test item, the string corresponding to some linguistic object (in double-quites), the description (often left unspecified, in order to get all possible parses) for the test linguistic object, expected number of solutions and comment.

Quick and compact introductions to installing and running TRALE, and writing TRALE grammars can be found in (Rosen, 2010b) and (Rosen, 2010a), respectively. For more details about TRALE's syntax, components and various software tools linked to it, the user's guide (Penn & Abdolhosseini, 2003a) is the only choice. (Penn *et al.*, 2003) represents a short guide to TRALE specific functionalities apart from ALE. TRALE reference manual (Penn & Abdolhosseini, 2003b) contains a specification of the data structures and algorithms of TRALE.

3. Modeling the syntax of Georgian

In this chapter, we model the syntax of the core part of Georgian in the bHPSG framework. The chapter is organized in a step-wise fashion. We gradually model some phenomena of the syntax and hence gradually develop the formal grammar for Georgian. Each stage of modeling starts with the analysis of the language phenomenon – considering examples covering the phenomenon and designing a system of rules describing it. Then we describe this system of rules in the HPSG formalism and integrate it with the already existing formal grammar, presenting revised components of the grammar. After the HPSG-based grammar is "written", we implement it in TRALE and explain some implementation issues. At the end of the modeling stage, we demonstrate the implemented grammar at work using the example.

We start the chapter by introducing the concepts of simple declarative sentence, declarative verb and logical case, since our approach to model the syntax adopts the methods of formal semantics introduced by (Pkhakadze, 2005) and treats these notions as fundamental.

The first steps in modeling the syntax of Georgian is to describe the simple declarative sentence and its constituents – the verb and its noun argument(s). We model the verb complementation and free word order on the phrase level. After that we discuss the feature of polypersonal agreement of the verb – the ability to encode the information about several persons on the syntactic level, and model its information in the formal grammar. We also model the pronoun and finish the verb complementation with modeling the polypersonal agreement.

We start modeling the noun phrase by adjunction of the noun by the adjective and the quantifier.[1] In this part we also adopt the formal analysis of adjectives and quantifiers introduced in (Pkhakadze, 2005).

The topic of adjunction of the noun is continued by describing the adjunction by possessive nouns and pronouns. We introduce the possessive case (contrary to the traditional Georgian grammar) and contrast it with the genitive case. For modeling different syntactic relations between adjuncts – adjectives, quantifiers, possessive nouns and pronouns, and the head noun we introduce the degree of specification.

We continue the discussion about possessive nouns and show that they represent just one of two roles of the noun in possessive. We contrast these two roles – possessive (i.e., adjunct) and complement role using examples and show how these two roles cause syntactic (and semantic) ambiguities of the surface structure of the noun phrase. Note that the formal analysis of the possessive adjuncts along with capturing and distinguishing possessive nouns from the complements in possessive are original contributions of this thesis. These contributions can be considered as a continuation of the formal research started by (Pkhakadze, 2005)

[1]In this way our formal grammar will cover the same range of phenomena as (Pkhakadze, 2005) and our implemented parser will parse the same range of sentences (and not only sentences) as systems (Abzianidze, 2008) and (Chikvinidze, 2010) based on (Pkhakadze, 2005) do.

in Georgian syntax-semantics.

We finish the chapter by giving the “hand-written” formal grammar modeling the syntax of the simple declarative sentences consisting of the verb and its complement noun phrases – the head possibly complemented and possibly modified by adjectives, quantifiers or possessives. The source code of the implemented TRALE grammar is also provided.

3.1 Simple declarative sentence and logical case

For modeling the syntax of Georgian, we base on the theory of simple declarative sentences and logical cases presented in (Pkhakadze, 2005) for Georgian.

We start with introducing the notion of simple declarative sentence and its key constituent – declarative verb, then continue with the notion of logical case naturally induced by former notions. At the end, the complete list of logical cases will be given and shown for each case its logicality. Note that these notions are the fundamental ones in the *logical grammar of the Georgian language*.[1]

3.1.1 Simple declarative sentence and the declarative verb

The approach we will use for the formal analysis of Georgian is mainly guided by semantics, specifically, by lexical semantics. This solution is justified from the fact that lexical semantics plays a crucial role in any grammar of Georgian, starting from morphology to semantics. While writing the formal grammar for Georgian, we build on the idea of K. Pkhakadze to view the Georgian declarative verb as a formal n-ary predicate with cases assigned to each position (Pkhakadze, 2005) (or (Pkhakadze, 2009a) in English). But the analysis of the declarative verb starts with the analysis of the Georgian simple declarative sentence. Below, we briefly explain the notion of simple declarative sentence and declarative verb in Georgian[2]. Also, for brevity, we often skip the word Georgian while talking about Georgian language.

In (Pkhakadze, 2005), the simple declarative sentence is defined as an *atomic*[3] formula of some mathematical language formalizing the *Georgian thinking language*.[4] A simple declarative sentence states some *atomic* and *complete* proposition via the predicate word on some noun phrases marked by cases and the stated proposition has a truth value. The unique predicate word (formally, predicate symbol) in the simple declarative sentence (formally, in an atomic formula) is (formally, is interpreted as) the Georgian declarative verb.[5] Therefore, the simple declarative sentence is a n-ary declarative verb applied to n number of terms, namely, noun phrases (hence, verbal forms like infinitives and gerunds are not allowed). Note that the verbal form can not be the declarative verb predicate since it can not form the simple declarative sentence.

After this, the following question arises: how is the arity of the Georgian declarative verb determined? The placeness of the declarative verb (simply, verb)

[1]The grammar is elaborated in the Open Institute of Georgian Language, Logic and Computer (GLLC). It represents the first Montagovian theory for Georgian.

[2]Further details about the research on Georgian done by K. Pkhakadze and the Open Institute of the Georgian Language, Logic and Computation (whose member the author of the thesis is) is beyond the thesis. The reader can find more details about the simple declarative sentence and verb in Georgian in (Pkhakadze, 2005) or in (Pkhakadze, 2008b), (Pkhakadze, 2008a), (Pkhakadze, 2009b), (Pkhakadze, 2009a) along with other related research topics.

[3]Atomic formula is commonly understood as in mathematical logic – a formula containing no logical connectives, thus having no subformulas. E.g., in first-order logic, an atomic formula is a n-ary predicate applied to n number of terms.

[4]The approach distinguishes three kind of languages, namely, *the Georgian written language*, *the Georgian spoken language* and *the Georgian thinking language* (Pkhakadze, 2009b).

[5]For simplicity, we will not distinguish the predicate symbol from its interpretation – the declarative verb.

is the number equal to the least number of different noun phrases necessary to *completely understand* the semantics of the proposition stated by the verb. Moreover, each place of the verb has its own domain. If we cannot define correctly the placeness of a particular verb and a domain of each place, this means that we do not understand completely the semantics of the verb (Pkhakadze, 2005). For better understanding of the above-mentioned notions, we explain them on several examples. We will use the superscript to indicates the placeness of verb predicates.

In (1), the predicate *gzavn-i-s*$^{(3)}$ states the proposition which talks only about three entities: a sender, an entity which is sent, an entity towards whom the sender is sending the thing. Note that the latter entity is only expressed by the semantics of the root of the verb (it is not explicitly expressed in the analytical gloss of the verb, but implicitly in the 'send' morpheme).

(1)	*ḳac-i*	*švil-tan*	*gzavn-i-s*$^{(3)}$	*c̣eril-s*
	Man-NOM	son/daughter-DAT-to[6]	3SING-send-PRS-IND-IPFV-3	letter-DAT

The man is sending[7] the letter to the son/daughter

Another good example is *v-u-ḳet-eb*$^{(3)}$ (2), where three entities should be considered in the proposition expressed by the verb, in order to completely understand the semantics of the verb. These entities are: the maker entity, the entity which is made by the maker and the entity for whom the maker is making the thing (i.e. the beneficent entity).

(2)	*ǰaḡl-s*	*v-u-ḳet-eb*$^{(3)}$	*saxl-s*
	Dog-DAT	1SING-3-make-PRS-IND-IPFV-3	house-DAT

I am making the house for the dog

Note that in order to understand the semantics of *ḳet-eb-a*[8] 'to make', two entities are required: the maker and the entity which is made; but the Georgian verb has the polypersonal property and there are inflectional rules that help to introduce new entities in the semantics of the inflected verb form. E.g. in (2), the beneficient entity is introduced by the *-u-* infix and that is why the semantics of the inflected verb becomes richer.

We continue providing examples and gradually increasing the placeness of verb predicates. (3) contains a 4-ary verb. Semantics of the non-finite verb *ga-gzavn-a* 'to send' talks about three entities, like (1). Example (3) represents its inflected verb form in such a way that the inflection introduces the beneficient entity (2nd person) in the semantics of the verb, like in (2). Thus the placeness of the verb predicate becomes four.

(3)	*g-i-gzavn-i*$^{(4)}$	c̣eril-s	deda-s-tan
	1SING-2SING-send-PRS-IND-IPFV-3	letter-DAT	mother-DAT-to

[6]We are still treating some postpositions in the traditional way assumed in the Georgian linguistics.

[7]Note that the verb in screeve 1 can express the present, general or future tense depending on the context. When the tense is not important for the discussion, we will show only one of the tenses in glosses, as in this case.

[8]More precisely, the *initial verbal semantic unit* (introduced in (Pkhakadze, 2004)). It is the semantic unit of the the non-finite verb *ḳet-eb-a* 'to make' and underlies to semantics of all inflected verb forms of the non-finite verb.

I am sending the letter to the mother for you

Note that the verb predicate *g-i-gzavn-i*$^{(4)}$ in (3) should not be confused with the 3-ary predicate with the same word form in (4). The verb *g-i-gzavn-i*$^{(3)}$ can be roughly considered as the version of *g-i-gzavn-i*$^{(4)}$ where the beneficent entity is identical to the addressee. Moreover, these verb predicates have different word forms in aorist (i.e., screeve 7): *ga-g-i-gzavn-e*$^{(4)}$ and *ga-mo-g-i-gzavn-e*$^{(3)}$. Note that the placeness of the verb predicate remains constant in the word forms inflected according to TAM grammatical features.[9]

(4) *g-i-gzavn-i*$^{(3)}$ *c̣eril-s*
1SING-2SING-send-PRS-IND-IPFV-3 letter-DAT
I am sending the letter to you

In the end, we exemplify the 5-ary verb predicate in (5). The semantics of the verb *ga-v-u-b-i*$^{(5)}$ implicitly states the proposition involving four entities: the agent entity performing the action, the patient entity which is stretched and two entities between which the patient is stretched. The extra fifth place (for the beneficent) is introduced by the *-u-* infix.

(5) *ḡob-idan xe-mde babua-s toḳ-i ga-v-u-b-i*$^{(5)}$
Fence-from tree-till grandfather-DAT rope-NOM PVB-1SING-3-stretch-between-PST-IND-PFV-3
I stretched the rope for the grandfather from the fence till the tree

Here we finish providing examples. More examples of the Georgian declarative verbs will be given when we discuss the notion of the logical case.

The reader can see some connections between the theta roles and theta relations (from Government and binding theory), and the approach given in (Pkhakadze, 2005). Both theories try to describe the relation between entities established by the verb, but the main difference is that the starting point of the former theory is the level of syntax and the one of the latter is semantics. More connections can be found between the earlier theory of C. J. Fillmore, called Case grammar and its further development – Frame semantics, and the theory of (Pkhakadze, 2005), as the theories are driven from the semantics of the verb. Unfortunately, there is no mention about the relation with those theories in (Pkhakadze, 2005) and its further discussion is beyond the scope of this thesis.

3.1.2 The notion of logical case

In the previous section, we introduced the notion of the Georgian simple declarative sentence and its core component – the Georgian declarative verb, also called the linguistic predicate (Pkhakadze, 2005), or simply verb predicate. Based on the simple declarative sentence and the linguistic predicate, (Pkhakadze, 2005) introduces the notion of the *logical case*.

While the linguistic predicate is characterized by the number of arguments, each of its argument is characterized by the syntactic domain (and not only by

[9]In other words, the verb forms differing from each other only in TAM grammatical features have the same number of places.

that). The initial syntactic domain is viewed as N-α, where N stands for the noun stem, followed by the delimiter (-) and α – the postposition morpheme. The linguistic predicate can be expressed formally as:

$$LingPred^{(n)}(N\text{-}\alpha_1, N\text{-}\alpha_2, \ldots, N\text{-}\alpha_n)$$

where the $N-\alpha_i$ expresses that the i-th argument of the predicate should be prefixed with α_i morpheme, otherwise the resulting formula (i.e., the simple declarative sentence) will be ill-formed (i.e., ungrammatical). Note that we assume that the order of the semantic roles for every linguistic predicate is fixed, thus the semantic role of each argument is unambiguously defined. We will call the ordered list of arguments of the linguistic predicate its semantic argument structure. The *logical case* stands for the N-α domain, and the α morpheme is the logical case marker. The different α morphemes result different logical cases.

For example, the linguistic predicate in (1) has the following semantic argument structure:

$$gzavnis^{(3)}(N\text{-}i, N\text{-}s, N\text{-}tan)$$

meaning that noun arguments should be marked with the following logical cases N-i, N-s and N-tan as it is in (1). The formula:

$$gzavnis^{(3)}(\underset{.}{k}ac\text{-}i,\ \underset{.}{c}eril\text{-}s,\ švil\text{-}tan)$$

corresponds to the Georgian simple declarative sentence given in (1).

After introducing the notion of the logical case, a question naturally arises: how many logical cases are there in Georgian? (Pkhakadze, 2005) shows that the number of logical cases is fourteen and each case is substantiated by a proof, providing an example of a Georgian simple declarative sentence built by the declarative verb which has this case in its semantic argument structure. In table 3.1, we give the complete list of the (fourteen) logical cases in Georgian.

#	Logical Case[10]	Name[11] of Case	Abbr.	English analog
1	N-*i*	Nominative	NOM	We, a box
2	N-*it*	Instrumental	INS	By hand, with knife
3	N-*is*	Genitive	GEN	Of him, of a box
4	N-*isḳen*	Orientative	ORI	Towards him
5	N-*idan*	Ablative1	ABL1	From/out of a box
6	N-*isgan*	Ablative2	ABL2	From us
7	N-*istvis*	Benefactive	BEN	For us
8	N-*ma*	Ergative	ERG	Man-ERG
9	N-*ši*	Inessive	INE	In a box
10	N-*s*	Dative	DAT	Us, to us
11	N-*ze*	Locative	LOC	On a box
12	N-*tan*	Comitative	COM	With us
13	N-*ad*	Adverbial	ADV	As/into a box
14	N-*amde*	Terminative	TER	Till/until/before a box

Table 3.1: The list of the Georgian logical cases.

The proof that for each inflected form a logical case is involved is given below with several examples of simple declarative sentences.

PROOF: A postpositional inflection is a logical case if there exists a verb predicate which has a word form in its semantic argument structure derived by this postpositional inflection. The following simple declarative sentences are built with verb predicates which provide the evidence for the fourteen logical cases:

gzavnis$^{(3)}$ *(**N-i**, **N-s**, **N-tan**)*
ḳac-i švil-tan gzavn-i-s$^{(3)}$ *ċeril-s*
The man-**NOM** is sending the letter-**DAT** to the son/daughter-**COM**

gaaba$^{(4)}$ *(**N-ma**, N-i, **N-idan**, **N-amde**)*
ḡob-idan saxl-amde biċ-ma toḳ-i ga-a-b-a$^{(4)}$
The boy-**ERG** stretched the rope-NOM from the fence-**ABL1** till the tree-**TER**

ešinia$^{(2)}$ *(N-s, **N-is**)*
ḳurdḡel-s ešin-i-a$^{(2)}$ *mgl-is*
The rabbit-DAT is afraid of the wolf-**GEN**

moixibla$^{(2)}$ *(N-i, **N-it**)*
kac-i kal-it mo-i-xibl-a$^{(2)}$
The man-NOM was charmed by the woman-**INS**

čavarda$^{(2)}$ *(N-i, **N-ši**)*
burt-i qut-ši ča[12]*-vard-a*$^{(2)}$
The ball-NOM fall in the box-**INE**

davarda$^{(2)}$ *(N-i, **N-ze**)*
burt-i magida-zi da[13]*-vard-a*$^{(2)}$
The ball-NOM fall on the table-**LOC**

šetrialda$^{(2)}$ *(N-i, **N-isḳen**)*
kac-i saxl-isḳen še-triald-a$^{(2)}$
The man-NOM turned towards the house-**ORI**

gadmoiḡo$^{(3)}$ *(N-ma, N-i, **N-isgan**)*
bavšv-ma čvev-eb-i mšobl-eb-isgan gadmo-iḡ-o$^{(3)}$
The child-ERG inherited the habits from the parents-**ABL2**

gadaakcia$^{(3)}$ *(N-ma, N-i, **N-ad**)*
ǩocna-m baqaq-i p̣rinc-ad gada-akc-i-a$^{(3)}$
The kiss-ERG transformed the frog-NOM into the the princes-**ADV**

miucia$^{(3)}$ *(N-s, N-i, **N-stvis**)*
deda-s vašl-i bavšvis-tvis mi-u-c-i-a$^{(3)}$
The mother-DAT has given the apple-NOM to the baby-**BEN** □

It is very likely that the complete list of the logical cases is represented only by these fourteen logical cases. In order to discover a new logical case, one has to find a Georgian verb whose semantic argument structure will have a word form not marked by the above-mentioned cases.

Note that the number of logical cases is twice greater than the number of

[12]The preverb adds the semantics about the target of the falling process to the verb ‘to fall’.
[13]The preverb adds the semantics about the target of the falling process to the verb ‘to fall’.

traditional cases in Georgian.

It is worthwhile to mention that two cases used in the traditional Georgian grammar (namely, vocative and genitive[14]) are not present in the list of logical cases. They are rejected for a simple reason – it is not possible to find a simple declarative sentence whose argument will be in vocative or in the traditional genitive case. Moreover, in contrast to the traditional Georgian cases, logical cases are qualified by the well-defined criteria based on the lexical semantics of the word. With the help of the criteria, it is possible to determine if some inflection of the noun is a logical case or not.

[14]The traditional genitive case is different from the logical genitive case. The difference will be revealed in one of the declension paradigms which we will discuss later.

3.2 Getting started with an HPSG-based grammar for Georgian

We start with modeling the syntax of simple declarative sentences in HPSG framework. First, we model the simple sentence with the noun arguments and then we extend it to the pronoun arguments. While the verb encodes the polypersonal information, we structure the lexical entry of the verb in such way to distinguish *implicit* and *explicit* arguments. This structuring allows an easy discovery for the verb saturation.

3.2.1 Complementation of verbs with nouns

The above theory about the Georgian simple declarative sentences, declarative verbs and logical cases represents the fundamentals of the formal grammar we are going to build for Georgian. We start building the grammar by modeling the core part of simple declarative[1] sentences. More precisely, the formal grammar will license only sentences which correspond to declarative verbal predicates with (partially or fully) complemented[2] semantic argument structures.

The lexicon of the initial grammar contains only verb forms and noun forms. The grammar will license not only sentences such as (1), corresponding to fully complemented verb predicates, but also sentences corresponding to partially complemented verbal predicates such as (6),

(6) *ḳac-i gzavn-i-s*$^{(3)}$ *c̣eril-s*
Man-NOM 3SING-send-PRS-IND-IPFV-3 letter-DAT
The man is sending the letter

where the addressee of the latter is not syntactically realized in the sentence.[3]

In the initial grammar, nouns include information about their logical case (henceforth simply "case"[4]), number and person categories, where the person category is always 3rd person.[5] The verbs carry information about their corresponding linguistic predicates, namely, the arity and the case category for each argument slot. The information will be encoded as a list value of the valency feature. This feature is also appropriate to nouns and its value is the empty list. Moreover, we add information about the number and person categories to each argument slot as the verb agrees with the noun in number and person.

The information about both word classes will be traditionally encoded in the HEAD feature of the syntactic category of the *word* and *phrase* types. Fig. 3.1 shows the main part of signature of the grammar. We assume that the most general type of the type hierarchy is *bot* (to be consistent with the *bot* type

[1] As we are mostly dealing with simple declarative sentences and declarative verbs, we often skip the word *declarative* while talking about them.

[2] Meaning that the argument slots of the verbal predicate are filled by some noun phrases

[3] In the formal grammar, we will be able to distinguish sentences corresponding to a verbal predicate with a partially and fully complemented semantic argument structure.

[4] Under "case", we do not mean those traditional seven cases any more, but the logical cases of (Pkhakadze, 2005)

[5] We introduce the grammatical category of person as we have intend to include pronouns in next versions of the grammar.

assumed in TRALE).

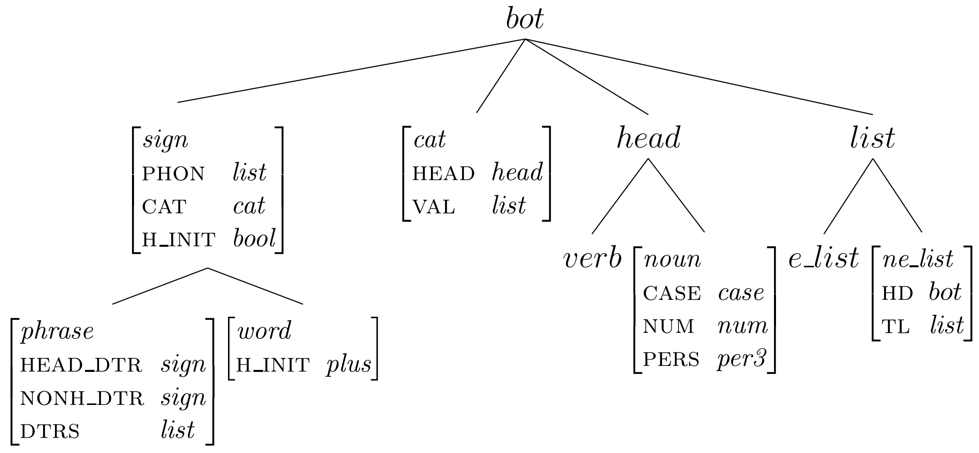


Figure 3.1: The first part of the signature of the initial grammar of Georgian

The *list* type represents the general list – a list of *bot*s. Details of the *boolean* type are depicted in Fig.3.2. The type stands for the boolean value and is directly subsumed by *bot*. The H_INIT feature has the boolean value. It serves to specify whether the leftmost daughter of the *phrase* is its head daughter or not. For the *word* type the value of the feature is maximally specified (i.e., resolved) to *plus*, assuming that each word is head initial (i.e., headed).[6] The *num* type stands for the number category. The *case* type is a type for logical cases and has all fourteen logical cases as its immediate subtypes.

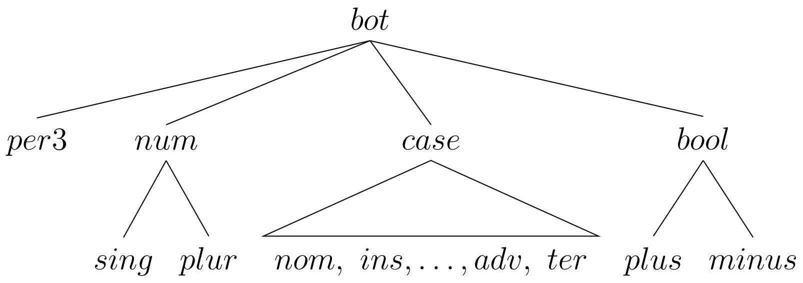


Figure 3.2: The second part of the signature of the initial grammar of Georgian

The sample lexical entries for the noun and the verb word classes are given in Fig.3.3 and Fig.3.4, respectively. The AVMs are labeled at the top with English translations. Other analytic information about the entry can be read from the AVMs.

The valency of the verb represents a list of *cats*[7] as the verb itself does not encode the information about the phonology of its arguments, but their syntactic and semantic properties. Note that the AVM corresponding to 'sends' is not type-resolved (the last two NUM features are not specified) because, in Georgian, the verb *gzavn-i-s* can be used with one or several sent and sendee entities; but in the

[6]The H_INIT feature is appropriate to *word* for the technical reasons – to avoid the unary phrase structure rules from WORDs to PHRASEs.

[7]Note that in the signature we just typed the VAL feature with the general list type.

[Man]

$$\left[\begin{array}{ll} \textit{word} & \\ \text{PHON} & \langle \textit{kaci} \rangle \\ \text{CAT} & \left[\begin{array}{ll} \textit{cat} & \\ \text{HEAD} & \left[\begin{array}{ll} \textit{noun} & \\ \text{CASE} & \textit{nom} \\ \text{NUM} & \textit{sing} \\ \text{PERS} & \textit{per3} \end{array}\right] \\ \text{VAL} & \langle\rangle \end{array}\right] \\ \text{H\_INIT} & \textit{plus} \end{array}\right]$$

Figure 3.3: A lexical entries for 'man' in initial grammar

complete context the AVM will be typed-resolved. If the AVM of some linguistic expression is not type-resolved it can be viewed as a underspecified representation of some linguistic expressions.

[Sends]

$$\left[\begin{array}{ll} \textit{word} & \\ \text{PHON} & \langle \textit{gzavnis} \rangle \\ \text{CAT} & \left[\begin{array}{ll} \textit{cat} & \\ \text{HEAD} & \textit{verb} \\ \text{VAL} & \left\langle \left[\begin{array}{ll} \textit{cat} & \\ \text{HEAD} & \left[\begin{array}{ll} \textit{noun} & \\ \text{CASE} & \textit{nom} \\ \text{NUM} & \textit{sing} \\ \text{PERS} & \textit{per3} \end{array}\right] \\ \text{VAL} & \langle\rangle \end{array}\right], \left[\begin{array}{ll} \textit{cat} & \\ \text{HEAD} & \left[\begin{array}{ll} \textit{noun} & \\ \text{CASE} & \textit{dat} \\ \text{NUM} & \textit{num} \\ \text{PERS} & \textit{per3} \end{array}\right] \\ \text{VAL} & \langle\rangle \end{array}\right], \left[\begin{array}{ll} \textit{cat} & \\ \text{HEAD} & \left[\begin{array}{ll} \textit{noun} & \\ \text{CASE} & \textit{com} \\ \text{NUM} & \textit{num} \\ \text{PERS} & \textit{per3} \end{array}\right] \\ \text{VAL} & \langle\rangle \end{array}\right] \right\rangle \end{array}\right] \\ \text{H\_INIT} & \textit{plus} \end{array}\right]$$

Figure 3.4: A lexical entry for 'sends' in initial grammar

In order to build grammatical phrases from lexical entries, we need to introduce phrase structure rules – to combine lexical entries and phrases into larger phrases, and principles – to license all and only grammatical expressions by constraining lexical entries and generated phrases.

In the HPSG theory, grammar principles and phrase structure rules (and also lexical entries) have the same format – all are expressed as constraints on relevant types or structures, usually in the shape of logical implications. Constraints corresponding to PS rules are called phrase schemata and are usually encoded as a disjunction with the type phrase as the antecedent; while, in TRALE, principles and phrase structure rules have different formats and functions. As our aim is to implement an HPSG grammar for Georgian, we will write principles and rules in the TRALE format. This will keep the grammar theory and its implementation close to each other and make it easy for the reader to understand.

In the initial version of the grammar, we use two phrase structure rules: Head-Initial Phrase (HIP) rule and Head-Final Phrase (HFP) rule. The AVM representation of the rules are given in Fig. 3.5 and Fig.3.6, respectively. Note that we use two types of arrows: thin arrows for phrase structure rules, and thick arrows for logical implications.

The rules simply says that the head expression should try to form the phrase first with all its right adjacent expressions and then with the left ones.

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{plus} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1} \end{bmatrix} \longrightarrow \boxed{0}\begin{bmatrix}\text{H\_INIT} & \textit{plus}\end{bmatrix} \quad \boxed{1}$$

Figure 3.5: The preliminary version of Head-Initial Phrase Rule

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{minus} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1} \end{bmatrix} \longrightarrow \boxed{1} \quad \boxed{0}$$

Figure 3.6: The preliminary version of Head-Final Phrase Rule

Along with the HIP and HFP rules, the initial grammar contains two principles: Head-Feature Principle (Fig.3.7) and Valency Principle (Fig.3.8).

$$\begin{bmatrix}\textit{phrase}\end{bmatrix} \Longrightarrow \begin{bmatrix} \text{CAT}|\text{HEAD} & \boxed{0} \\ \text{HEAD\_DTR}|\text{CAT}|\text{HEAD} & \boxed{0} \end{bmatrix}$$

Figure 3.7: Head-Feature Principle

In Valency Principle, $\ominus$ stands for the deletion operation. It operates on the non-empty list L and some type t and $L \ominus t$ returns the list R different from L only in t. Note that the operation is defined if t is in L, otherwise it is undefined. Moreover, the operation can return several results if there are several ts in L.

$$\begin{bmatrix}\textit{phrase}\end{bmatrix} \Longrightarrow \begin{bmatrix} \text{CAT}|\text{VAL} & \boxed{1} \ominus \boxed{0} \\ \text{HEAD\_DTR}|\text{CAT}|\text{VAL} & \boxed{1} \\ \text{NONH\_DTR}|\text{CAT} & \boxed{0} \end{bmatrix}$$

Figure 3.8: Valency Principle

Below we give the implementation of $\ominus$ operation as a functional description `del` in TRALE (List.3.1). TRALE's functional notation `del(Y,X)=Z` corresponds to the infix notation $X \ominus\ Y = Z$ of the delete operation, where the `del` function is directly defined by the `del/3` predicate. In the implementation, we use delayed version of the deletion operation. Which means that `when/2` delays execution until some event is witnessed. After the condition is satisfied, it calls the procedures defined in the second position. For more details about `when/2` and its usage we refer the reader to (Penn & Abdolhosseini, 2003a, pp. 44).

To demonstrate how the initial grammar works, let us consider the simple sentence (7).

(7) *student-i ḳitx-ul-ob-s çign-s*
Student-NOM read-PRS-IND-IPFV-3-SING-3 book-DAT
The student reads the book

Our grammar licenses (7), and gives the following AVM parse tree[8] for it (Fig.3.9).

[8]In general, there is no need for the parse tree to represent the final unification, since the unification is an information combining operator and the final unification contains all necessary information about the sentence structure; but the parse tree with AVMs as nodes is often used for its visual power.

Listing 3.1: The functional description Delete

```
fun del(+,+,-).
del(X,Y,Z) if
   when( (Y=ne_list;
          Z=(e_list;ne_list)),
        undelayed_del(X,Y,Z)).

undelayed_del(El,(hd:El,tl:L),L) if true.
undelayed_del(El,(hd:H,tl:T1),(hd:H,tl:T2)) if
          del(El,T1,T2).
```

Note that according to HIP and HFP rules [Reads][9] first combines with [Book] and then with [Student] and, at the same time, these combinations are licensed by Valency and HF principles.
The current grammar also licenses the free word order among the verb and its arguments (i.e., complements). We will call the current version of the Georgian grammar GEOGRAM1. The source code of the grammar is attached to the thesis as appendix A.

3.2.2 Verb complementation – polypersonal agreement

In the current section we are going to introduce the pronouns and the verb forms encoding polypersonal agreement – rich information on several arguments at the same time. Also, we will add the information about tense to the verb signs.

Before we started modifying the grammar, let us discuss the polypersonal property of the verbs once again. In previous grammar, the verb simply had the nouns in its argument structure and encoded the information about their case, number and person (always specified as *per3*) features. Now some verbs have to assign different person features to the arguments. Let us see on examples how the verbs does this.

In (8), the verb corresponding to a unary predicate marks its single argument with the first person.

(8) *v-arseb-ob*
1SING-exist-PRS-IND-IPFV
I exist

Note that the semantics of (8) remains the same if the argument (i.e., the first person pronoun) of the verb is syntactically realized. In (9), the verb encodes the person features (and the number features too) for both arguments and the syntactic realization of the arguments is optional. This is also an example that a single verb form can represent a simple sentence.

(9) *g-xaṭ-av*
2SING-draw-PRS-IND-IPFV-1SING
I am drawing you

[9]As a shorthand notation, we denote the AVM of *some phrase* by [*some, phrase*].

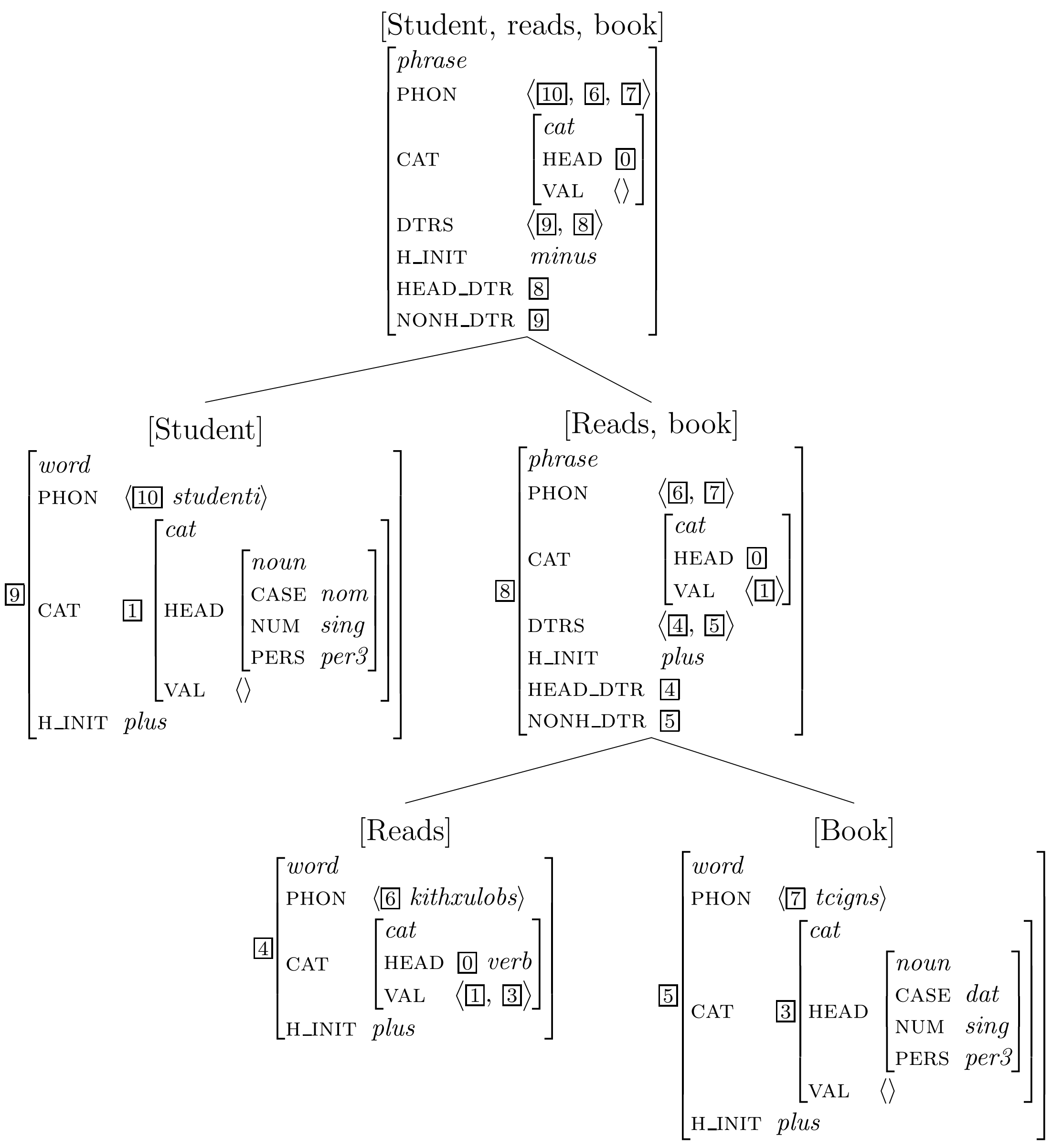


Figure 3.9: the AVM parse tree for the simple sentence

The verb in (10), encodes person features for all its arguments, but the number feature for the agent argument. Hence, the number feature for arguments except the agent can be specified by syntactic realizations of the arguments, e.g. (11).

(10) *v-u-xaṭ-av*
1SING-3-draw-PRS-IND-IPFV-3
I am drawing it for her

Note that the theme of the action 'draw' is not marked (namely, its person feature) explicitly in the morphology of the verb, but it has no choice to have other than the third person feature, since encoding the person features except third person is done by different verb forms, e.g. (9) where the theme argument of 'draw' is the 2nd person.

(11) *vašl-s* *v-u-xaṭ-av*
Apple-DAT 1SING-3-draw-PRS-IND-IPFV-3

I am drawing the apple for her

Moreover, there are no verb forms in Georgian encoding the information (12) in its morphology. To express events like these, some arguments (not specified by the verb form) have to be syntactically realized as it is exemplified in (13).[10]

(12) *1SING-1/2-draw-PRS-IND-IPFV-3
I am drawing myself/you for her

(13)

(13)	*v-u-xaṭ-av*	*čem-s/šen-s*	*tav-s*
	1SING-3-draw-PRS-IND-IPFV-3	my/your-DAT	head-DAT
	I am drawing myself/you for her		

We will call the arguments, which the verb provides with the person feature, the *explicit* arguments.[11]

Now let us see the verbs which do not encode any person feature information for some arguments. (14) exemplifies such a kind of verb form.

(14)	*m-ešin-ia*	*šen-i*
	1-SING-afraid-of-PRS-IND-IPFV	you-GEN
	I am afraid of you	

The following grammatical sentence (15), shows that the verb ‘afraid of’ does not provide its second argument with the person feature (but does provide it with the case). There are several kinds of such verbs.

(15)	*m-ešin-i-a*	*mgl-is*
	1-SING-afraid-of-PRS-IND-IPFV	wolf-GEN
	I am afraid of a wolf	

We call such kind of arguments the *implicit* arguments. Moreover, we observe that the features of implicit arguments of the verb remain constant in inflected forms of the verb, e.g. (16)-(19).

(16)	*ešin-i-a*	*šen-i/mgl-is*
	3SING-afraid-of-PRS-IND-IPFV	you/wolf-GEN
	She is afraid of a wolf	
(17)	*ešin-o-d-a*	*šen-i/mgl-is*
	3SING-afraid-of-PST-IND-IPFV	you/wolf-GEN
	She was afraid of a wolf	
(18)	*še-ešin-d-a*	*šen-i/mgl-is*
	3SING-afraid-of-PST-IND-PFV	you/wolf-GEN
	She got frightened of the wolf	
(19)	*še-ešin-d-eb-a*	*šen-i/mgl-is*
	3SING-afraid-of-FUT-IND-IPFV	you/wolf-GEN
	She will get frightened of the wolf	

Therefore, the collection of implicit arguments can be viewed as a kind of vertical generalization of lexical entries. In the grammar, we are going to distinguish

[10]Here we give just one of the sentences expressing this semantics.
[11]Although the number feature of an explicit argument can be left unspecified by the verb.

implicit arguments from explicit ones in verbal lexical entries. This will be useful from the lexical and syntactic point of view.[12]

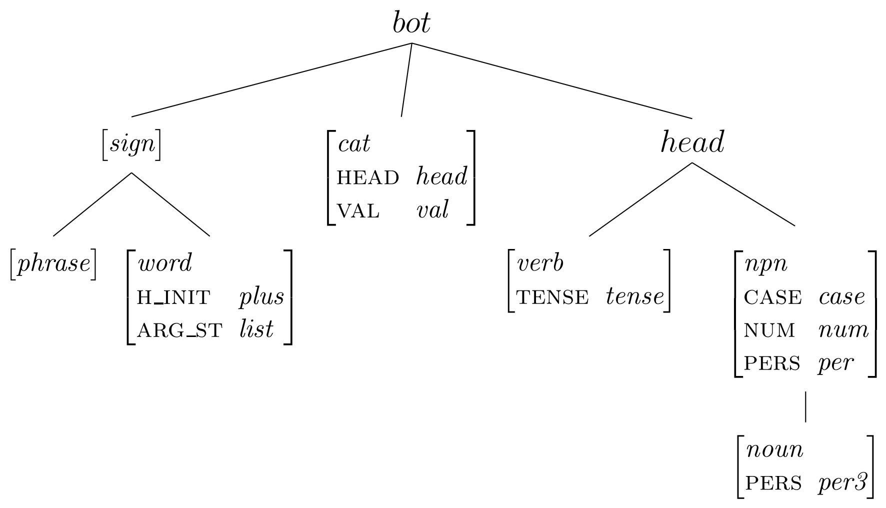


Figure 3.10: The first part of the updated signature

In order to introduce new categories in the grammar, we add some new types to the signature of GEOGRAM1. The changes in the signature are depicted in Fig. 3.10 and Fig. 3.11. We introduce *npn* to cover the nouns and pronouns. *npn* has the feature PERS unspecified, in contrast to its subtype *noun*. A new feature TENSE of type *tense* is appropriate to *verb*, where *tense* has four subtypes corresponding to the present (*prs*), past continuous (i.e., progressive) (*psc*), past (*pst*) and future (*fut*) tenses.

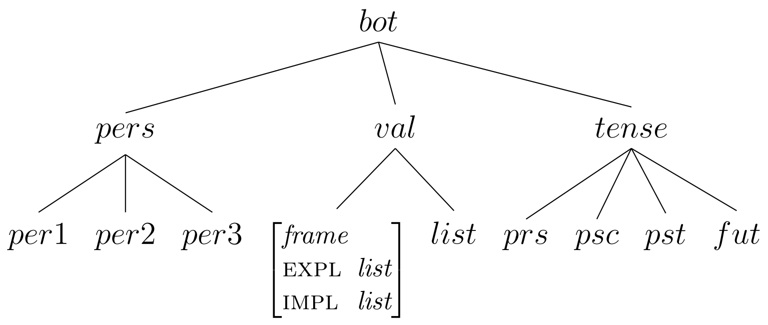


Figure 3.11: The second part of the updated signature

In order to structure the information about implicit and explicit arguments of the verb, the type of the VAL feature is updated to *val*, which has two subtypes *frame* and *list* (Fig. 3.11). The *frame* type is intended to structure the argument structure of the verb into two *lists* – lists of explicit and implicit arguments. Note that we keep the original argument structure in the feature ARG_ST attached to *word*. This will facilitate to talk about the semantic argument structure of the

[12]We believe that the difference between explicit and implicit arguments plays some role in the phrase order of a sentence. However, we are not exploring phrase order in this work (assuming that it is free), mainly due to the lack of resources necessary for research of this kind.

verb.[13] At this moment we simply consider other word classes than verbs to have empty semantic argument structure.

We also update the theory of the grammar,[14] namely, macros, functional descriptions and principles. In Valency Principle we have to consider changes we did in the valency structure of the verb. The revised version of Valency Principle is given in Fig.3.12. Now the verb complement should complement one of the 'slots' either from the explicit or implicit list.

$$\begin{bmatrix}\textit{phrase}\end{bmatrix} \Longrightarrow \begin{bmatrix} \text{CAT}|\text{VAL} & \begin{bmatrix}\text{EXPL} & \boxed{1}\ominus\boxed{0}\\ \text{IMPL} & \boxed{2}\end{bmatrix}\\ \text{HEAD\_DTR}|\text{CAT}|\text{VAL} & \begin{bmatrix}\text{EXPL} & \boxed{1}\\ \text{IMPL} & \boxed{2}\end{bmatrix}\\ \text{NONH\_DTR}|\text{CAT} & \boxed{0}\end{bmatrix} \bigvee \begin{bmatrix} \text{CAT}|\text{VAL} & \begin{bmatrix}\text{EXPL} & \boxed{1}\\ \text{IMPL} & \boxed{2}\ominus\boxed{0}\end{bmatrix}\\ \text{HEAD\_DTR}|\text{CAT}|\text{VAL} & \begin{bmatrix}\text{EXPL} & \boxed{1}\\ \text{IMPL} & \boxed{2}\end{bmatrix}\\ \text{NONH\_DTR}|\text{CAT} & \boxed{0}\end{bmatrix}$$

Figure 3.12: Revised Valency Principle

In the lexical entry of the verb, the semantic argument structure is defined in terms of explicit and implicit argument structures. Namely, the value of ARG_ST is the list resulted by appending the list values of EXPL and IMPL (Fig.3.13).

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \langle \textit{gagigzavne} \rangle\\ \text{ARG\_ST} & \boxed{0}\oplus\boxed{1}\\ \text{CAT} & \begin{bmatrix} \textit{cat} & \\ \text{HEAD} & \begin{bmatrix}\textit{verb} & \\ \text{TENSE} & \textit{pst}\end{bmatrix}\\ \text{VAL} & \begin{bmatrix} \textit{frame} & \\ \text{EXPL} & \boxed{0}\left\langle \begin{bmatrix}\textit{cat} & \\ \text{HEAD} & \begin{bmatrix}\textit{npn} & \\ \text{CASE} & \textit{erg}\\ \text{NUM} & \textit{sing}\\ \text{PERS} & \textit{per1}\end{bmatrix}\\ \text{VAL} & \langle\rangle\end{bmatrix}, \begin{bmatrix}\textit{cat} & \\ \text{HEAD} & \begin{bmatrix}\textit{npn} & \\ \text{CASE} & \textit{dat}\\ \text{NUM} & \textit{sing}\\ \text{PERS} & \textit{per2}\end{bmatrix}\\ \text{VAL} & \langle\rangle\end{bmatrix}, \begin{bmatrix}\textit{cat} & \\ \text{HEAD} & \begin{bmatrix}\textit{npn} & \\ \text{CASE} & \textit{nom}\\ \text{NUM} & \textit{num}\\ \text{PERS} & \textit{per3}\end{bmatrix}\\ \text{VAL} & \langle\rangle\end{bmatrix} \right\rangle\\ \text{IMPL} & \boxed{1}\left\langle \begin{bmatrix}\textit{cat} & \\ \text{HEAD} & \begin{bmatrix}\textit{npn} & \\ \text{CASE} & \textit{com}\\ \text{NUM} & \textit{num}\\ \text{PERS} & \textit{per3}\end{bmatrix}\\ \text{VAL} & \langle\rangle\end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3.13: A lexical entry for the verb form *ga-g-i-gzavn-e* (I-sent-it-for-you)

The append operator denoted by the symbol ⊕ takes two lists as the input and returns the list consisting with the same elements in the same order as the input lists. The functional description `append` is the implementation of the append ⊕ operator in TRALE (3.2).

[13]This approach is similar to the one in (Sag *et al.*, 2003).

[14]As long we do not differentiate HPSG-based grammar and the implemented TRALE grammar in the thesis, under the theory we mean all parts of the grammar except the signature, the lexicon and the test suite.

Listing 3.2: The functional description Append

```
fun append(+,+,-).
append(X,Y,Z) if
   when( (X=(e_list;ne_list);
          Z=(e_list;ne_list)),
        undelayed_append(X,Y,Z) ).

undelayed_append(e_list , L, L) if true.
undelayed_append((hd:H, tl:T1), L, (hd:H, tl:T2)) if
          append(T1, L, T2).
```

In addition, we could also optimize the phrase structure rules but we leave this for the next version of the grammar.[15]

To see how the grammar works, we show the AVM parse tree of the sentence (20) in Fig. 3.14, where the branches are labeled by H denoting head daughters. Note that the verbal semantic argument structure is completed with arguments, in other words, the verb is saturated; though some arguments are not realized in the sentence as overt constituents. The saturation of the verb can be checked easily by checking whether the value of the IMPL feature is empty or not.

(20)	*ga-g-i-gzavn-e*	*c̣eril-eb-i*	*sṭudenṭ-eb-tan*
	1SING-2SING-3-send-PST-IND-PFV	letters-NOM	students-COM
	I sent for you letters to students		

We will refer to the current version of the grammar as GEOGRAM2. The source code of GEOGRAM2 can be found in Appendix B.

[15]In the current grammar, any phrase is headed by the verb, so we could specify this information in the phrase structure rules to make parsing more efficient.

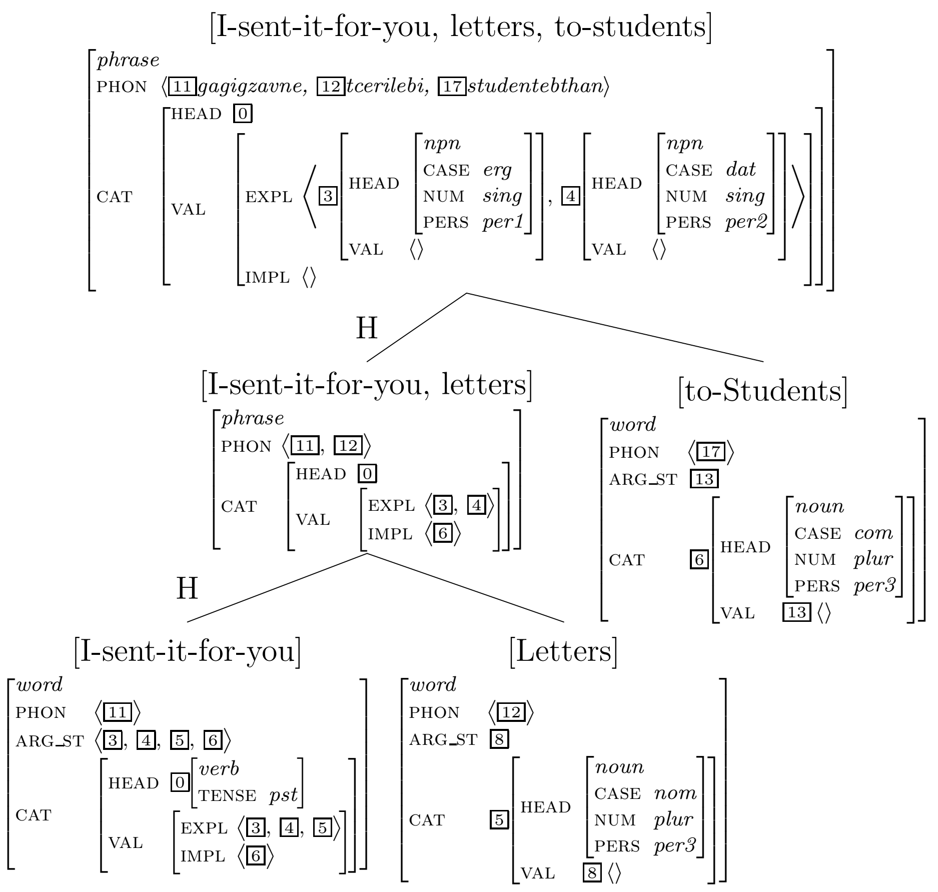

[I-sent-it-for-you, letters, to-students]
phrase
PHON ⟨11 gagigzavne, 12 tcerilebi, 17 studentebthan⟩
CAT HEAD 0
VAL EXPL ⟨3 HEAD npn CASE erg NUM sing PERS per1 VAL ⟨⟩, 4 HEAD npn CASE dat NUM sing PERS per2 VAL ⟨⟩⟩
IMPL ⟨⟩
H
[I-sent-it-for-you, letters]
phrase
PHON ⟨11, 12⟩
CAT HEAD 0
VAL EXPL ⟨3, 4⟩ IMPL ⟨6⟩
[to-Students]
word
PHON ⟨17⟩
ARG_ST 13
CAT 6 HEAD noun CASE com NUM plur PERS per3
VAL 13 ⟨⟩
H
[I-sent-it-for-you]
word
PHON ⟨11⟩
ARG_ST ⟨3, 4, 5, 6⟩
CAT HEAD 0 verb TENSE pst
VAL EXPL ⟨3, 4, 5⟩ IMPL ⟨6⟩
[Letters]
word
PHON ⟨12⟩
ARG_ST 8
CAT 5 HEAD noun CASE nom NUM plur PERS per3
VAL 8 ⟨⟩

Figure 3.14: The AVM parse tree of the saturated sentence

3.3 The noun phrase – Adjunction and complementation of the noun

In the previous section 3.2, we modeled the syntax of simple declarative sentences – the verb complemented by nouns and pronouns, and implemented in the formal grammar GEOGRAM2. In the current section, we extend our formal grammar by modeling the syntax of noun phrases, hence make the grammar to parse simple sentences with noun phrases.

We begin with modeling the adjunction of the noun. First, we model a noun phrase consisting of attribute modifiers (i.e., adjectives) followed by the head noun, then we deal with quantifiers and model quantified noun phrases. We finish the part on adjuncts by modeling noun phrases with possessives (possessive nouns and pronouns).

We start complementation of the noun by contrasting noun complements with possessive nouns (i.e., several readings of some noun phrases vs. the alleged left-branching). Then we model the complementation in the grammar. At the end of the section, we present the implementation of the modified grammar.

3.3.1 Adjunction by adjectives and quantifiers

First, we show the similarity between adjectives and quantifiers while agreeing with (but differently modifying) the head noun and then model the adjunction.

All these will be done according to the theory of (Pkhakadze, 2005).

Adjunction by adjectives

Like in (almost) all languages, in Georgian the adjective is also one of the common adjuncts of the noun. Hence, it is natural to start modeling noun phrases with nouns modified by adjectives. Note that in this part, we will talk about modifier adjectives which should not be confused with nominalized adjectives – nouns derived from adjectives which really behave as nouns in syntax.

In Georgian, the adjective agrees with the noun only in case.[1] (Pkhakadze, 2005) defines four logical cases for the adjective, based on the logical cases for nouns. We distinguish these two kinds of logical cases as *adjunct logical cases* and *argument logical cases*,[2] but in some unambiguous context we still refer to them as logical cases or just cases. Adjunct logical cases are the inflections which the adjective takes while declining the noun phrase (the adjective modifying the head noun) according to argument logical cases. Simply saying, we call adjunct logical cases to those inflections of adjuncts which are required for the agreement with the noun marked by some argument logical case.

The list of adjunct cases[3] along with the set of compatible argument cases are given in Table 3.2.

Adjunct logical case	Compatible argument logical cases
NOM	NOM, INS, GEN, ORI, ABL1, ABL2, BEN
ERG	ERG
DAT	DAT, INE, LOC, COM, ADV, TER
CNST	All argument logical cases

Table 3.2: The list of adjunct logical cases

Note that adjunct cases are divided into two groups, where the CNST[4] case is appropriate only to the adjective whose word form remains constant during declension (hence, this is the only case it can have); the NOM, ERG and DAT[5] cases are appropriate to the rest of adjectives – changing their word form during declension.[6]

As in (almost) all languages, several adjectives can modify the same noun in Georgian, but an adjective does not modify a pronoun.[7]

[1]Unlike in Slavic languages, where the adjective agrees with the noun in number and gender categories too.

[2]As these cases are assigned to arguments of the verb predicate.

[3]The adjunct case system, differs from the one used in the traditional grammar in the vocative case.

[4]It roughly corresponds to case A in (Pkhakadze, 2005), where case and POS are integrated in the type A.

[5]They correspond to the cases of A-*i*, A-*ma* and A- in (Pkhakadze, 2005), respectively.

[6]Note that these two different kinds of adjectives are easily distinguished by their word form, which agrees with the noun in the nominal case. Namely, if it ends with -*i*, then the adjective has three word forms, otherwise it has a constant word form. We will revisit this issue when we write lexical rules.

[7]More details about this issue are given in lexical rules section.

Adjunction by quantifiers

Quantifiers behave as adjuncts of the noun in the syntax, but they have a stronger semantic impact than other adjuncts. Like adjectives, quantifiers also agree with nouns in case. Moreover, their case inflection coincides with the one for adjectives – as in adjectives, there are two kinds of quantifiers: where the word form either remains constant, or changes during declension. Table 3.2 describes the agreement between the case of the quantifier and the case of the head as well as between that of the adjective and the head.

Unlike adjectives, quantifiers are sensitive to the number category of the head. Namely, they cannot specify a plural noun, e.g. the noun phrase (21) is ungrammatical.

(21) **qvela* *student-eb-i*
All-CNST Students-NOM
All students[8]

Like in English, in Georgian the order between the quantifiers, adjectives and nouns in the noun phrase are described by the regular language Q?A*N – head noun preceded by possibly zero or more number of adjectives prefixed with the optional quantifier.

Revision of the formal grammar

For modeling adjunction of the noun, we introduce new types in the signature of the grammar. First, we revise the *case* type (Fig. 3.15), we make it more general to cover more than just argument logical case (*arg_c*). Now it subsumes adjunct logical case (*adj_c*), where *adj_c*'s subtypes correspond to the adjunct cases of Table 3.2. The subtypes of *arg_c* are organized according to Table 3.2 that makes it easy to capture the agreement between *nom_c*, *erg_c*, *dat_c* and *case_i*, *erg*, *case_*, respectively.[9]

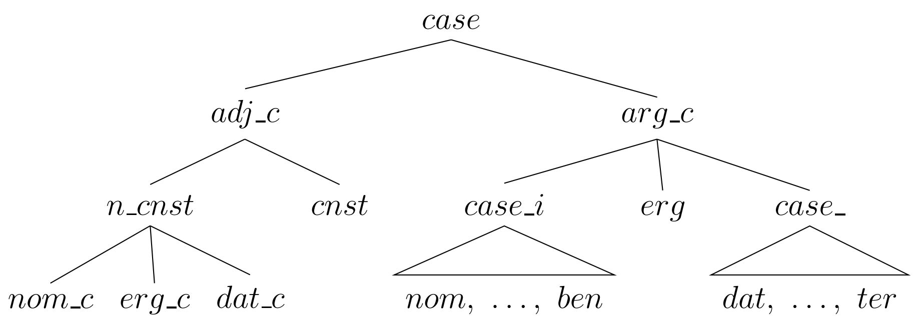


Figure 3.15: *case* type and its subtypes

We also add the adjective type (*adj*) and the quantifier type (*qnt*) with other subtypes to *head* (Fig. 3.16). Note that *adjunct* has an appropriate feature MOD,

[8]Unlike in English.

[9]There could be a more compact way of organizing the case type hierarchy – identifying *n_cnst* with *arg_c* and their agreed immediate subtypes, and *adj_c* with *case*. But this solution would give fourteen type-resolved TFSs for the adjective in CNST case, even though it is a single linguistic object. This will be the case for the possessive nouns as you will see in the later sections.

which will include the information about the modified category in the *ne_list* type. In Georgian, agreement between the adjunct and the head is restricted to case, hence there is no need of appropriating a special agreement feature (commonly, AGR) for the head type.

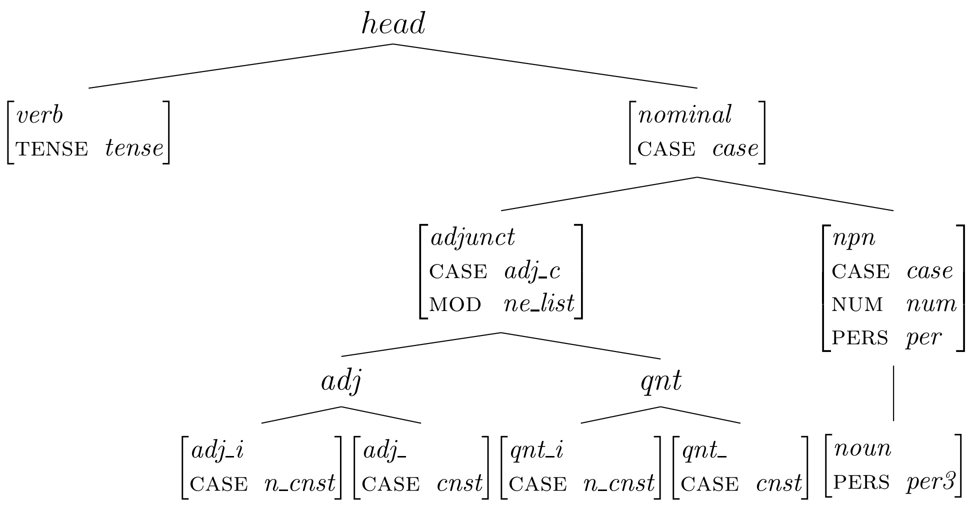


Figure 3.16: *head* type and its subtypes

The maximal type having the CASE feature of type *case* is *nominal*. The *adjunct* and *npn* subtypes partition the head of a noun phrase and its possible adjuncts and assign different types for the CASE feature. The subtypes of *adjunct* encode the information about adjectival and quantifier adjuncts and their subtypes with the appropriate types for the CASE feature.

Some changes affect also the valency type (*val*). Namely, we add a new subtype *spec* with the SPEC feature of type *bool*. The value of SPEC will denote whether the noun phrase is quantified or not. Using the information from SPEC the grammar does not licenses any adjunction (by the quantifier or adjective) of already quantified noun.

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{minus} \\ \text{CAT}|\text{VAL}|\text{SPEC} & \boxed{5} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1} \end{bmatrix} \longrightarrow \boxed{1}\begin{bmatrix} \textit{word} & \\ \text{CAT}|\text{HEAD} & \boxed{3}\begin{bmatrix} \textit{adjunct} & \\ \text{CASE} & \boxed{7} \\ \text{MOD} & \langle \boxed{4} \rangle \end{bmatrix} \end{bmatrix} \; \boxed{0}\begin{bmatrix} \text{CAT} & \boxed{4}\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \boxed{6} \end{bmatrix} \\ \text{VAL}|\text{SPEC} & \textit{minus} \end{bmatrix} \end{bmatrix}$$

WHERE, IF $\boxed{3} = \textit{adj}$ THEN $\boxed{5} = \textit{minus}$, OTHERWISE *plus*, AND $\boxed{7}$ IS COMPATIBLE WITH $\boxed{6}$

Figure 3.17: The preliminary version of AdjN Rule

In the theory of the grammar, we introduce a new phrase structure rule – adjunct-noun (AdjN) rule (Fig. 3.17). The rule licenses a phrase consisting of an adjunct word followed by an unspecified[10] head noun, such that the case of the adjunct has to be compatible (in the sense of Table 3.2) with the case of its modified category and the modified category of the adjunct should be consistent with the category of the noun. The value of SPEC of the licensed phrase depends

[10]This prevents the ungrammatical combinations like the adjective or the quantifier preceding the quantified head noun.

on the adjunct type. In the lexicon, the SPEC feature of the pronouns is specified to *plus*, in order to avoid their adjunction by an adjective or quantifier.

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{plus} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1} \end{bmatrix} \longrightarrow \boxed{0}\begin{bmatrix} \text{CAT}|\text{HEAD} & \textit{verb} \\ \text{H\_INIT} & \textit{plus} \end{bmatrix} \boxed{1}$$

Figure 3.18: Verb-Initial Phrase Rule (revised Head-Initial Phrase Rule)

Since free word order is only between the arguments of the verb, we modify the HIP and HFP rules and make them specific for the verb.[11] Namely we revise and rename the rules as Verb-Initial Phrase and Verb-Final Phrase rules (Fig. 3.18 and Fig. 3.19), respectively.

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{minus} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1} \end{bmatrix} \longrightarrow \boxed{1} \quad \boxed{0}\begin{bmatrix} \text{CAT}|\text{HEAD} & \textit{verb} \end{bmatrix}$$

Figure 3.19: Verb-Final Phrase Rule (revised Head-Final Phrase Rule)

Moreover, we revise and rename the Valency Principle as the Verb Valency Principle as the consequent of the principle is satisfied only by the verb phrase (Fig. 3.20).

$$\begin{bmatrix} \textit{phrase} & \\ \text{CAT}|\text{HEAD} & \textit{verb} \end{bmatrix} \Longrightarrow \begin{bmatrix} \text{CAT}|\text{VAL} & \begin{bmatrix} \text{EXPL} & \boxed{1} \ominus \boxed{0} \\ \text{IMPL} & \boxed{2} \end{bmatrix} \\ \text{HEAD\_DTR}|\text{CAT}|\text{VAL} & \begin{bmatrix} \text{EXPL} & \boxed{1} \\ \text{IMPL} & \boxed{2} \end{bmatrix} \\ \text{NONH\_DTR}|\text{CAT} & \boxed{0} \end{bmatrix} \bigvee \begin{bmatrix} \text{CAT}|\text{VAL} & \begin{bmatrix} \text{EXPL} & \boxed{1} \\ \text{IMPL} & \boxed{2} \ominus \boxed{0} \end{bmatrix} \\ \text{HEAD\_DTR}|\text{CAT}|\text{VAL} & \begin{bmatrix} \text{EXPL} & \boxed{1} \\ \text{IMPL} & \boxed{2} \end{bmatrix} \\ \text{NONH\_DTR}|\text{CAT} & \boxed{0} \end{bmatrix}$$

Figure 3.20: Verb Valency Principle (revised Valency Principle)

The current version of our grammar can license verbs complemented with pronouns and noun phrases – some adjuncts (adjectives and quantifiers) followed by a head noun. To demonstrate how the current formal grammar works, we parse the sentence (22) and give its simplified AVM parse tree in Fig. 3.21.

(22)	*qvela*	*c̣eril-i*	*ga-g-i-gzavn-e*	*bejit*
	All-CNST	letter-NOM	1SING-2SING-3-send-PST-IND-PFV	diligent-DAT
	sṭudenṭ-eb-tan			
	students-COM			

I sent all letters to diligent students for you

3.3.2 Possessive nouns and pronouns

We continue modeling the adjunction of the noun – specifying the head noun by possessives. We start with contrasting the logical genitive and the traditional

[11]We also make the parser more efficient this way.

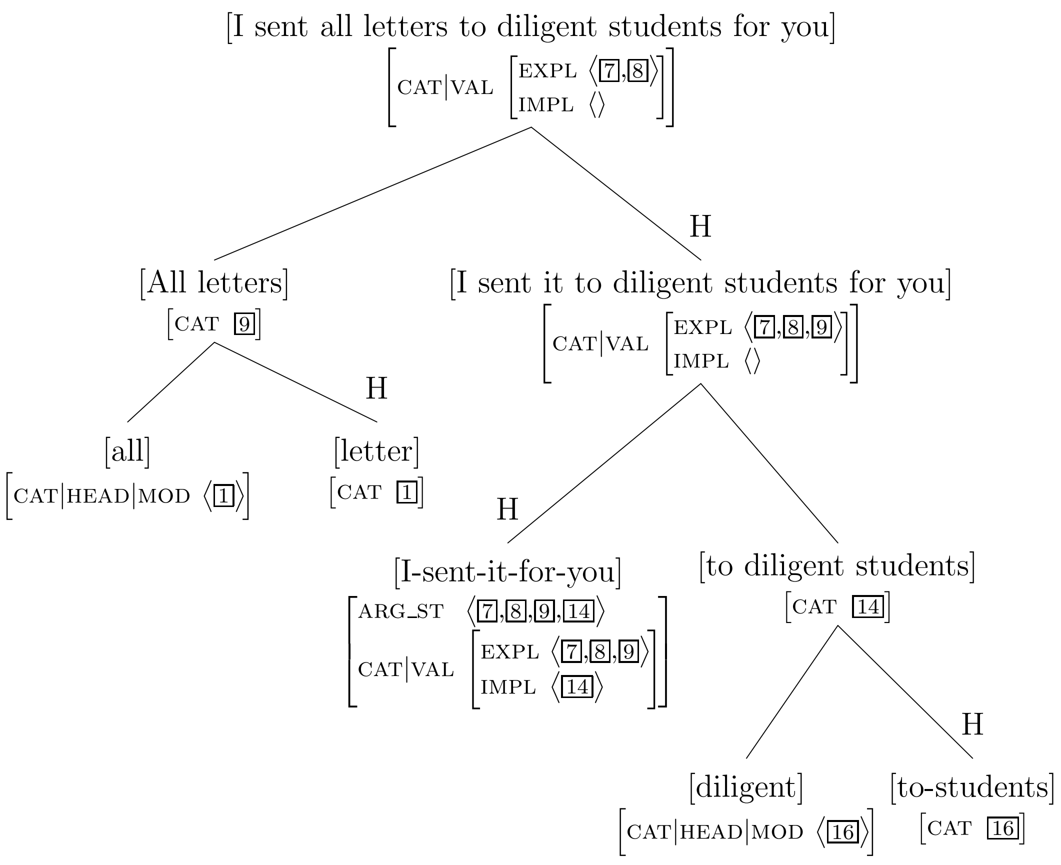

[I sent all letters to diligent students for you]
CAT|VAL EXPL ⟨7,8⟩ IMPL ⟨⟩
H
[All letters]
CAT 9
[I sent it to diligent students for you]
CAT|VAL EXPL ⟨7,8,9⟩ IMPL ⟨⟩
H
[all]
CAT|HEAD|MOD ⟨1⟩
[letter]
CAT 1
H
[I-sent-it-for-you]
ARG_ST ⟨7,8,9,14⟩
CAT|VAL EXPL ⟨7,8,9⟩ IMPL ⟨14⟩
[to diligent students]
CAT 14
H
[diligent]
CAT|HEAD|MOD ⟨16⟩
[to-students]
CAT 16

Figure 3.21: The AVM parse tree of the sentence (22)

genitive cases, then introduce a new case – possessive case to model the possessive noun. After analyzing how the possessive noun specifies the head, we introduce the degrees of specification and describe its change during all grammatical adjunctions. At the end, we do the same analysis for possessive pronouns and according to it, extend the degrees of specification.

Possessive nouns

In general, possessive nouns are nouns marked with either the genitive or the possessive case (depending on the language). They are used to express the possessive relation between concepts.

According to the traditional Georgian grammar the possessive noun is marked by the genitive case and there is no possessive case in Georgian. One of the results of (Abzianidze, 2008) was showing the difference between the logical genitive and the traditional genitive case.[12] While modeling the syntax of Georgian, we go further and state that there are both possessive and genitive cases in Georgian, and the former is used to mark the possessive nouns and the latter corresponds to the logical genitive case (*N-is*, according to (Pkhakadze, 2005)). As we showed in Section 3.1.2, the genitive case (as we analyzed the inflection of the word form *N-is*) is a logical case. For most nouns, the word form marked by the possessive case is phonologically identical to the one marked by the genitive case. But for example, the noun *gogo* ‘girl’ distinguishes these two word forms. In (23), the

[12]Originally, the difference between logical case *N-is* and the traditional genitive case.

word form *gogo-si* corresponds to the (logical) genitive case (since the word form represents an implicit argument of the verbal predicate).

(23)

m-erid-eb-a	*gogo-si*
1SING-shy-of	girl-GEN

I am shy of a girl

The possessive form of ‘girl’ is given in (24). The nouns which distinguish these two inflections are the nouns ending with *-u* or *-o* in nominative, or representing the proper names (e.g. human names).

(24)

gogo-s	*c̣ign-i*
girl-POSS	book-NOM

The girl’s book

The different word forms for these syntactically and semantically different roles make us distinguish possessive and genitive cases.

In our formal grammar, we mark possessive nouns by possessive case and the case is different from the logical (i.e., argument) cases. While the noun representing an entity being possessed declines according to the logical cases, the noun in the possessive case does not change its word form (i.e., stays in possessive case). By this kind of behavior, possessive nouns resemble adjectives and quantifiers, whose form also remains constant during the declension of the modified head. Hence, from the point of agreement, the noun in possessive agrees with the noun in all logical cases. There is nothing more than this agreement between the possessive and the possessed nouns.

Possessive nouns specify the head differently than quantifiers and adjectives. Namely, they can modify a noun phrase consisting of adjectives and quantifiers and followed by the noun (i.e., noun phrases which are licensed by the current version of the formal grammar) as in (25).

(25)

gogo-s	*qvela*	*lamaz-i*	*surat-i*
girl-POSS	all-CNST	beautiful-NOM	picture-NOM

The girl’s all beautiful pictures

But after the possession of the noun phrase is expressed – the noun phrase is modified by the possessive noun, the head of the noun phrase is not accessible (for agreement) to adjectives and quantifiers any more, as exemplified in (26).

(26)

**lamaz-ma*	*kal-is*	*mankana-m*
Beautiful-ERG	woman-POSS	car-ERG

Woman’s beautiful car

There are two options to fix (26). One option is that the adjective modifies the possessive noun and results in a possessive noun phrase (27). Note that the adjectival and quantifier adjuncts have to be marked by the nominative case to modify the possessive noun (i.e., the possessive noun behaves as if it was marked by nominative in adjunction).

(27)

lamaz-i	*kal-is*	*mankana-m*
Beautiful-NOM	woman-POSS	car-ERG

(Beautiful woman's) car

Another option is to swap the possessive noun and the adjunct (28). Note that the semantics of these two grammatical phrases is different and the syntactic structures (shown by grouping) are also different.

(28)	*kal-is*	*lamaz-ma*	*mankana-m*
	Woman-POSS	beautiful-ERG	car-ERG
	Woman's (beautiful car)		

Also the possessed head is not accessible to the other possessive noun. Though the sentence remains grammatical (as the possessive noun modifies the possessive noun), syntactic structure (and semantics) of the phrase is different as the possession does not apply to the head, but to its possessive modifier (29).

(29)	*prezidenṭ-is*	*kal-is*	*mankana*
	President-POSS	woman-POSS	car-NOM
	(President's woman's) car		

To model the noun phrases correctly, the simple boolean value (as it is the type of SPEC feature of the noun category) is not sufficient any more to express the specification degree (i.e., status) of the head. In Table 3.3, we express how adjectives, quantifiers and possessive nouns change the different specification degrees of the head.

Type of adjunct	Spec-value of head	Resulting spec-value
Adjective	unspecified	→ attributed
Adjective	attributed	→ attributed
Quantifier	unspecified	→ quantified
Quantifier	attributed	→ quantified
Poss. noun	unspecified	→ possessed
Poss. noun	attributed	→ possessed
Poss. noun	quantified	→ possessed

Table 3.3: Possible specification values for the head noun

Each row of Table 3.3 describes all grammatical adjunction processes of the head noun, modified by adjectives, quantifiers and possessive nouns. E.g., if the quantifier adjunct modifies the unspecified[13] or attributed head noun, then the head's specification value becomes 'quantified'. Adjunction – quantifier modifying the possessed head noun – is not presented in the table as it is grammatically impossible (as exemplified in (29)).

The rules of specification change (Table 3.3) play a key role in the determination of grammatical adjunctions in the noun phrase. We will add more rules to these above after discussing possessive pronouns.

[13]This specification value is for the noun lexical entries – noun categories which are not modified.

Possessive pronouns

Possessive pronouns have almost the same syntactic and semantic functions as possessive nouns do. In contrast to possessive nouns, they encode the information about the category of person and agree with the modified head in case. They also reveal the richer nature of the specifying function than the possessive nouns do. We will discuss all these points below.

The possessive pronoun has the person, number and case categories. It agrees with the head in case like the adjective and the quantifier do (having three logical cases as shown in Table 3.2, section 3.3.1). Note that the genitive word form of the pronoun should not be confused with the nominative word form on the possessive pronoun. Though they are phonologically the same, they reveal syntactic and semantic functions different from those of nouns.

Unlike the possessive noun, the possessive pronoun can modify the head along with the quantifier. Moreover, both orders are permissible in the noun phrase, as shown in (30) and (31).

(30) *čem-ma or-ma megobar-ma*
My-ERG two-ERG friend-ERG
My two friends

(31) *or-ma čem-ma megobar-ma*
Two-ERG my-ERG friend-ERG
Two friends of mine

Taking into account the effect of the possessive pronouns on the specification value of the head, we update Table 3.3 to Table 3.4. Note that the adjective is a 'mild' specifier according to other adjuncts – after an adjective it is still possible to specify the head by other adjuncts.

Type of adjunct	Spec-value of head	Resulting spec-value
Adjective	unspecified	→ attributed
Adjective	attributed	→ attributed
Quantifier	unspecified	→ quantified
Quantifier	attributed	→ quantified
Quantifier	pro. possessed	→ poss+quant
Poss. noun	unspecified	→ noun possessed
Poss. noun	attributed	→ noun possessed
Poss. noun	quantified	→ noun possessed
Poss. pron.	unspecified	→ pro. possessed
Poss. pron.	attributed	→ pro. possessed
Poss. pron.	quantified	→ poss+quant

Table 3.4: Possible specification values for the head noun

The rules given in Table 3.4 will be used in the formal grammar and in its implementations. Based on this rules we will license all and only grammatical noun phrases.

3.3.3 Noun complementation and several readings of noun phrases

We begin with revealing the complementation relation, different from possession, between the head noun and noun in possessive. Then we see how the complementation of the noun causes several syntactic (and semantic) readings in some noun phrases, in contrast to the alleged left-branching readings. We analyze conditions when the noun complementation is possible and how it affects the specification of the head. At the end, making some conclusions about the semantics of complement nouns.

Adjunct nouns vs complement nouns in noun phrases

In the previous section we discussed possessives – possessive nouns and possessive pronouns, and analyzed the possessive relation between the noun in possessive and the head noun. Here, we are going to show another kind of relation, different from the possessive relation. Note that we use the expression – noun in possessive – to refer to the noun which is marked by the possessive case and is not necessarily the possessive noun – the noun syntactically serving as the adjunct (namely, as the determiner).

Let us consider the example below (32).[14] Based on the analysis done in Section 3.3.2, we can describe (32) as a noun phrase consisting of a head and an adjunct – a quantifier followed by a possessive noun.

(32)	*qvela*	*ḳutx-is*	*mbrǰanebel-i*
	All-CNST	region-POSS	lord-NOM

The lord of all regions

But there is also a different syntactic (and semantic) reading of the phrase, where the quantifier is applied to the head of the phrase rather than to the possessive noun (33).[15] This reading talks about all lords of a region (33).

(33)	*qvela*	*ḳutx-is*	*mbrǰanebel-i*
	All-CNST	region-POSS	lord-NOM

All lords of a region

This kind of reading is more obvious in the example (34). Where nobody will interpret (34) as a governor of all sates (opposed to the reading of (32)).

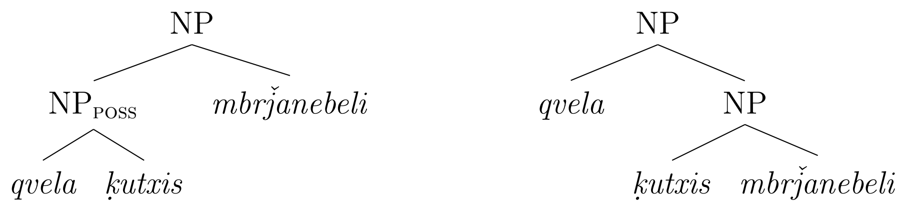
(34)	*qvela*	*šṭaṭ-is*	*gubernaṭor-i*
	All-CNST	state-POSS	governor-NOM

All state governors

Syntactic structures of these two readings are given in Fig. 3.22, where only the reading of (32) has the left-branching structure.

[14]By the way, the phrase is very popular in Georgian tales and history and refers to the lord of all areas/regions of some country.

[15]Analyzing the semantics of (33) and considering the two semantic readings – lords of the same region (i.e., the existential quantifier with 'wide' scope) or lords of some regions (i.e., the universal quantifier with 'wide' scope), is beyond this thesis.



(32) (33)

Figure 3.22: Two syntactic readings of the noun phrase

The reason for these two syntactic (and semantic) readings of the noun phrase is that there can be two kinds of syntactic relations between the noun in the possessive case and the head noun, namely a possessive relation – where the adjunct is a possessive noun and serves as a determiner and, a complement relation – where the noun in possessive is the complement (as opposed to the adjunct). The possessive relation expresses the possession of one entity by another, but the complement relation corresponds to the generic relation – complement relation, rather than just possession.

To make a rough parallel with English, English noun-modifying adjectives correspond to Georgian nouns in the possessive case establishing the complement relation with the head noun. To illustrate the point, let us consider (35), representing one of two reading of the surface structure in Georgian. The reading is specified by the parenthesis expressing the tree structure of the phrase. In the example, the noun in possessive (*avṭobus-is*) is a complement of the head noun (*gačereba*) and the complement does not refer to the particular bus but to any bus (semantically, a variable defined over the domain of buses). This relation in English is done by, so-called, noun modifiers (namely, by *bus*).

(35) *qvela* (*avṭobus-is gačereba*)
All-CNST bus-POSS stop-NOM
All bus stops

The second reading of the sentence is exemplified in (36)– talking about the particular stop where all buses stop. This kind of concept is expressed in different way in English, but in Georgian, the noun[16] in possessive also does it.

(36) (*qvela avṭobus-is*) *gačereba*
All-CNST bus-POSS stop-NOM
The stop of all buses

These two different functionalities – complementation and adjunction (namely, determination), expressed by the same word form – the word form of the noun marked by the possessive case, causes several syntactic (and semantic) readings of noun phrases. This phenomenon is easily resolved in human communication by taking into account the context of the communication.

Note that if we first quantify the head in (36), then we get (37) which refers to the different concept than (35) and (36).

[16]For shorthand, we refer to the noun phrase with its head marked in possessive as the noun in possessive.

(37)	*avṭobus-is*	*qvela*	*gačereba*
	Bus-POSS	all-CNST	stop-NOM

All stops of the bus

Also there are no two readings any more and the noun in possessive behaves as the determiner – all stops are the stops of the particular bus (semantically, a constant form the domain of buses, in contrast to the different nature of the same word in (35)). We can conclude from this example that the head after quantification loses its ability to take complements.

To conclude, we discovered another syntactic behavior of the noun marked by the possessive case. Namely, except being the adjunct of the head, it can also serve as a complement to the same head. These two roles cause several readings of the surface structure, opposed to the alleged left-branching structures which is specific only for the adjunct (namely, the determiner) nature. Moreover, these functionally different behaviors have different semantics. The complement noun is understood as a variable (i.e., a universal constant) over the domain of entities expressed by the noun and the determiner noun is a constant entity from this domain. The latter statement is another evidence of semantic diversity of the Georgian noun declared in (Pkhakadze, 2008b) and (Pkhakadze, 2009a).

Noun complementation

After discovering the new kind of syntactic relation between the noun marked by the possessive case and the head noun, we have to find out how the complement noun changes the head category and in which syntactic constructions the noun complementation is possible.

After some analysis of the noun phrases, we find that the complement does not affect the specification value of the head. This property is exemplified in (38), where both phrases – with or without the complement (enclosed in parenthesis) of the head – are grammatical.

(38)	*qvela*	*gabrazebul-i*	*(šṭaṭ-is)*	*gubernaṭor-i*
	All-NOM	angry-NOM	(state-POSS)	governor-NOM

All angry (state) governors

The complementation did not change the specification value of the head, but what are the admissible specification values for the head that it can take complements? The answer to the question is that the complementation of the head is possible only if the head is unspecified, otherwise the relation between the noun in possessive and the head will have the possessive nature. In (39), we show a noun phrase consisting of a noun in possessive followed by a specified head.

(39)	*(qvela)*	*šṭaṭ-is*	*gabrazebul-i*	*gubernaṭor-i*
	(All-NOM)	state-POSS	angry-NOM	governor-NOM

The angry governor of (all states) a state

Without any other details it is difficult to detect the type of the relation between the noun in possessive (*šṭaṭ*) and head (*gubernaṭor-i*). After we quantify the noun phrase by ‘all’, it does not quantify the head of the phrase but the noun in possessive. This shows that before the quantification there was a possessive relation in the noun phrase and it made the head inaccessible for the quantification (see Table 3.4). Therefore the early 'mild' specification (namely, by the adjective) forbids further complementation of the head. Like adjectives, specification by other adjuncts also prevents the complementation. The example for the quantifier 'blocking' the noun complementation is given in (37), where due to 'blocking' the surface structure is unambiguous.

Another issue that needs to be analyzed is – what kind of noun or noun phrase can be the noun complement? Note that answering such kind of questions is not easy and requires empirical checking on deeply annotated data. Despite this, after some analysis we ensure that the noun complement can be specified as attributed or quantified.[17] The corresponding examples in the same order are given in (40) and (41), where the complements are enclosed by parentheses.

(40)	(*p̣ersonalur-i*	*ḳomp̣iuṭer-is*)	*nac̣il-i*
	(Personal-NOM	computer-POSS)	part-NOM
	A part (of the personal computer)		
(41)	(*(xut-i*	*švil-is*)	*deda*
	(Five-NOM	son/daughter-POSS)	mother-NOM
	A mother (of five sons/daughters)		

Although the lack of resources for Georgian does not allow us to make further substantial statements about noun complements, we will encode the information about complements in lexical entries, therefore the grammar theory will not be affected by this lacuna.

At the end, we want to emphasize that there is no related study of Georgian noun phrases and noun complementation known to us. The researches done on the complementation in Georgian were more oriented on the complementation of the verb than the noun, like (Vamling, 1989).

3.3.4 The grammar and its implementation

We begin modeling the syntax of noun phrases in the grammar based on the analysis done in the previous sections. While revising the formal grammar, at some technical points we give the source code of its implementation. In the end, we present the complete code of the implemented TRALE grammar and demonstrate its parsing power.

Signature

We make changes in the signature after introducing possessive nouns and pronouns. Note that both share some features with adjuncts, nouns and pronouns. Hence, we organize the hierarchy in such a way that possessives inherit appropriate features from these categories. The revised *nominal* type of the signature with its subtype hierarchy is given in Fig. 3.23 (as a reminder, *nominal* is the direct subtype of *head* along with *verb*).

[17]Not all kind of quantifications are available. E.g. the universal quantification makes the head unusable for complementation, since the complement noun should be interpreted as a universal constant over the domain expressed by the head.

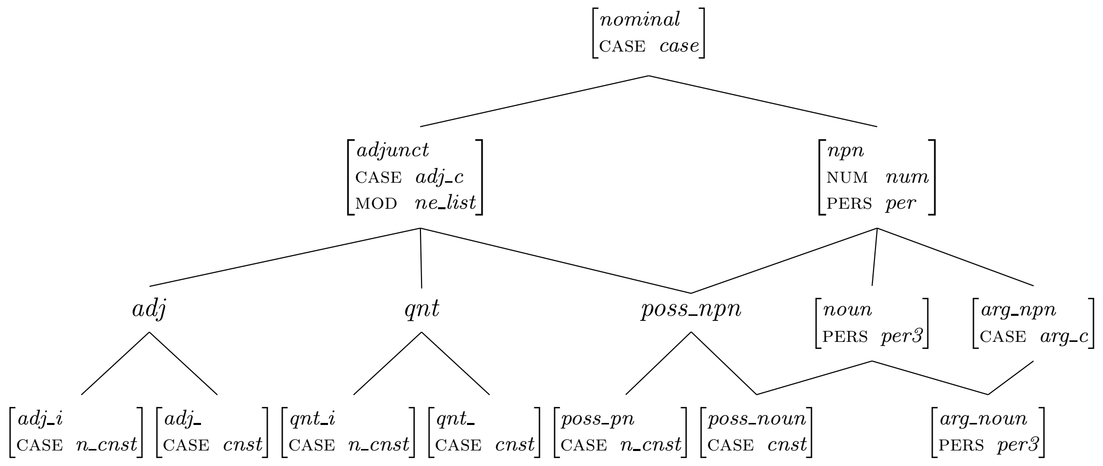


Figure 3.23: *nominal* type and its subtype hierarchy

We introduce five new types (*poss_npn*, *poss_pn*, *poss_noun*, *arg_npn*, *poss_noun*) in the subtype hierarchy subsumed by the *nominal* type. The noun/pronoun (*npn*) is partitioned into tree direct subtypes – *poss_npn* representing possessives; *noun* still remains as a subtype of *npn* and stands for the noun; *arg_npn* corresponding to nouns/pronouns which are candidates for the arguments of the verbal predicate, hence the argument case is appropriate to it. At the same time *poss_npn* is a subtype of adjuncts (*adjunct*) since possessives can syntactically behave like adjuncts. They inherit all features from adjuncts and nouns/pronouns. Note that *poss_npn* has the same case type (*adj_c*) as adjuncts. *poss_npn* has subtypes *poss_pn* and *poss_noun* representing possessive pronouns and nouns, respectively. Moreover, *poss_pn* has the same case type *n_cnst* as the adjectives and quantifiers with variable case. The case of possessive nouns is not the (logical) genitive case, but the possessive case. We do not introduce any new case type for the possessive case, but we mark the possessive nouns with the constant case (CNST). This kind of marking works flawlessly in the grammar theory as the possessive nouns have the same 'constant' agreement with the head like adjectives and quantifiers with the CNST case. Except of *poss_npn. poss_noun* is also a subtype of *noun. arg_noun* is a subtype of *arg_npn* and *noun* as the argument nouns are nouns and candidates for arguments at the same time.

A new feature is introduced at the *comps* valency type, namely, the COMP feature which encodes information about the syntactic category of the complements in the *list* type (Fig. 3.24). The *e_list* type is one of the subtypes of *val* and it is intended as a valency value for those syntactic categories which do not take any complements.

In order to express those various specification degrees the head noun can obtain in noun phrases, we revise the *spec* type and make it more structured. The *unspec* type is the specification value that corresponds to the unspecified sate (this state is important in the syntax as the head noun can be complemented only when it is unspecified). The 'opposite' type of *unspec* is *specif*, which can be specified to its several subtypes expressing different degrees of specification. The *attrib* type is for the attributed specification. *deter* stands for the quantified or possessed (by nouns or pronouns) specifications. The *quant* type stands for the quantified status and, *n_poss* and *pn_poss* for the possession by the noun

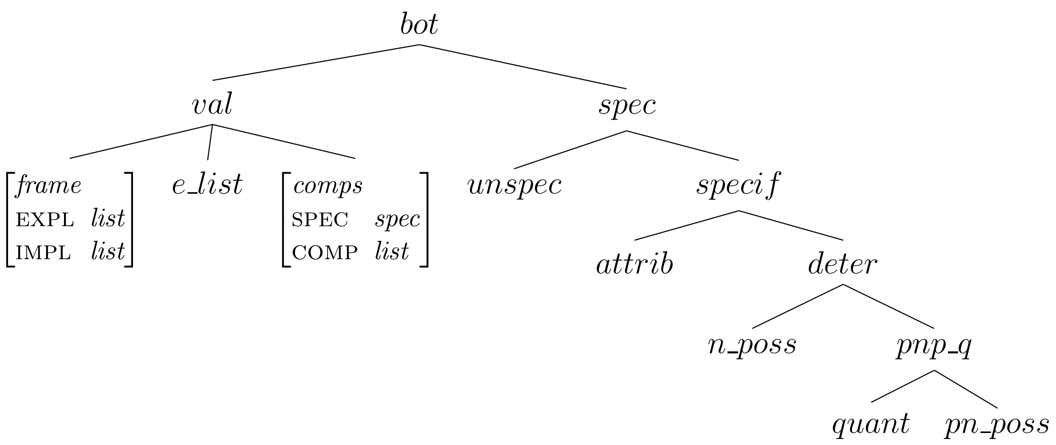


Figure 3.24: The *val* and *spec* types and their subtypes

and the pronoun, respectively. *pnp_q* is the type value of SPEC when the head noun is quantified and possessed by the pronoun at the same time.[18] All these specification types are recall of the specification statuses given in Table 3.4.

Theory

In the theory of the grammar, we revise the adjunction of the noun phrase by adding the possessive adjunction and then introduce the new rule – the noun complementation rule.

In order to consider the possessives in the noun adjunction, we revise the AdjN Rule (Fig. 3.25). The valency value of the phrase is revised by adding COMP feature with *e_list* as its value. This is due to the fact that after adjunction the head complementation is not possible (section 3.3.3). Hence, we specify the complement value as the empty list. Also, note that the adjunct constituent can be of type *word* or *phrase*, since the possessive adjunct can be a noun phrase.

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{minus} \\ \text{CAT}|\text{VAL} & \begin{bmatrix}\text{SPEC} & \boxed{3}\\ \text{COMP} & \textit{e\_list}\end{bmatrix} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1}\end{bmatrix} \longrightarrow \boxed{1}\begin{bmatrix}\text{CAT}|\text{HEAD} & \boxed{5}\begin{bmatrix}\textit{adjunct} & \\ \text{CASE} & \boxed{7} \\ \text{MOD} & \langle\boxed{4}\rangle\end{bmatrix}\end{bmatrix} \quad \boxed{0}\begin{bmatrix}\text{CAT} & \boxed{4}\begin{bmatrix}\text{HEAD} & \begin{bmatrix}\textit{noun} & \\ \text{CASE} & \boxed{6}\end{bmatrix} \\ \text{VAL}|\text{SPEC} & \boxed{2}\end{bmatrix}\end{bmatrix}$$

WHERE, SPECIFY($\boxed{5}$, $\boxed{2}$, $\boxed{3}$) AND COMPATIBLECASES($\boxed{7}$, $\boxed{6}$)

Figure 3.25: The final version of AdjN Rule

For the rule application[19] there are two necessary conditions to be met. The first condition is that the *head* type of the adjunct and the values of the SPEC features of the head and the resulting phrase should satisfy the ternary predicate SPECIFY defined according to Table 3.4. The second conditions remains the same

[18]This kind of structure of the specification type reflects the real situation in the noun phrase and it is smoothly implemented in TRALE grammar (namely, as Prolog's definite clauses).

[19]In the HPSG formalism, this would be satisfaction (rather than application) of the principle.

– it makes sure that the case of the adjunct is compatible with the one of the head. The binary predicate CompatibleCases is defined according to Table 3.2.

Since the order of the complement and the head is fixed in the noun phrase, we introduce the rule to model the noun complementation in the grammar. The Complement-Noun Rule (CNR) is exemplified in Fig. 3.26 (in the format of TRALE grammar).

$$\begin{bmatrix} \textit{phrase} & \\ \text{H\_INIT} & \textit{minus} \\ \text{CAT}|\text{VAL} & \begin{bmatrix}\text{SPEC} & \textit{unspec}\\ \text{COMP} & \textit{e\_list}\end{bmatrix} \\ \text{HEAD\_DTR} & \boxed{0} \\ \text{NONH\_DTR} & \boxed{1}\end{bmatrix} \longrightarrow \boxed{1}\begin{bmatrix}\text{CAT} & \boxed{2}\end{bmatrix} \quad \boxed{0}\begin{bmatrix}\text{CAT} & \begin{bmatrix}\textit{word} & \textit{head} & \textit{noun}\\ \text{VAL}|\text{COMP} & \langle\boxed{2}\rangle & \end{bmatrix}\end{bmatrix}$$

Figure 3.26: The final version of Complement-Noun Rule

The reason for the simplicity of the rule is that the information about the syntactic category of the complement is encoded in the lexical entry – in the value of COMP. For example, the case feature is encoded in the complement list and there is no need for an explicit constraint on it. Moreover, it is not necessary to ensure that the head constituent is unspecified, because the rule does not license the complementation of specified heads since AdjN rule sets the value of COMP to the empty list after the adjunction (i.e., specification). The complement list of the resulted phrase becomes the empty list. There is no need to constrain the head to be *word* as the noun phrase complementation is not licensed by the rule (due to AdjN rule assigning empty complement list to the licensed phrases).

The the remaining rules and principles of the theory are left unchanged.

Implementation

We start with discussing some technical issues of the grammar implementation. Then we give the complete source code of the implemented TRALE grammar and in the end, we show its parsing abilities.

In the AdjN rule of the formal grammar, we used two predicates – the ternary predicate Specify encoding the rules of Table 3.4 and the binary predicate CompatibleCase representing Table 3.2. The following predicates are implemented as clauses under the functional descriptions. The implementation of the predicate Specify as the `specify` functional description is given in List 3.3.

Listing 3.3: The functional description `specify`

```
fun specify(+,+,-).
specify((S,specif), unspec, S) if true.
specify((S,specif), attrib, S) if true.
specify(n_poss, quant, n_poss) if true.
specify(pn_poss, quant, pnp_q) if true.
specify(quant, pn_poss, pnp_q) if true.
```

The function takes any two types and if these types satisfy any clause then outputs the third argument type of that clause. The clauses are written in such a way that the output of the function is always deterministic (i.e., not more than one output value). For example, the first two clauses simply express the relation that unspecified and attributed specification degrees are always overridden by the specification functionality[20] of any adjunct (Table 3.4).

Listing 3.4: The functional description `mod_case`

```
fun mod_case(+,-).
mod_case(erg_c, erg)     if true.
mod_case(dat_c, case_)   if true.
mod_case(cnst, case)     if true.
mod_case(nom_c, cnst_i) if true.
```

The predicate CompatibleCase is also implemented as a functional description – `mod_case` (List 3.4). The function takes a type as its input and returns the compatible case type if it exists. Note that the *cnst_i* type is a direct supertype of *cnst* and *case_i*. The reason of introducing this type is rather technical and with the help of it we achieve smooth agreement between *case_i* and *cnst* – needed for the adjunction of possessive nouns. `mod_case` is used in macros of adjectives and quantifiers to automatically define the case type for the modified category based on the case type of the adjunct.

It is worthwhile to mention the technical reason for adding the *not_deter* type which represents the 'opposite' (i.e., complement) type of *deter*. The type is used for the specification value of noun complements. We avoid use of TRALE's inequality (`=\=`) for the negation of the type, because during the testing TRALE licenses the object even if it violates the inequality constraint and the grammar writer has to manually check if the inequity is satisfied or not.

In the signature, there are three subtypes introduced for *phrase*, namely, *ch_phrase* (the complement-head phrase), *hc_phrase* (the head-complement phrase) and *ah_phrase* (the adjunct-head phrase). With this new types it is easier to analyze the several resulted parse trees of the same linguistic expressions (i.e., surface structure), since they encode the dominance relation of daughters in the label of the mother node.

The complete source code of the implemented TRALE grammar along with the lexicon and test suite is presented in appendix C.

Demonstration

To see how the TRALE grammar works, we let it parse the sentence (42), including an ambiguous noun phrase.

(42)	*mo-m-ç̣on-s*	*čem-i*	*paṭara*	*bavšv-is*	*rol-i*
	1SING-like-PRS-IND-IPFV-3	my-NOM	little-CNST	child-POSS	role-NOM
	I like my little child's role				

All three output parse trees for (42) are given in Fig. 3.27, Fig. 3.28, Fig. 3.29.

[20]We avoid to give adjunct types as the first argument to `specify` since the type hierarchy of adjuncts is already defined and its direct use in clauses of `specify` would require introduction of some redundant unwanted subtypes of *head*.

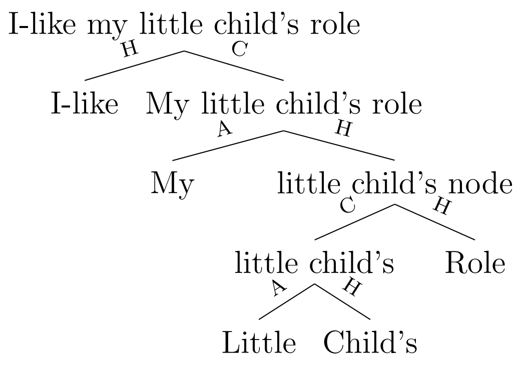


Figure 3.27: The first reading – I like my role of a little child

In the first reading, 'little child's' is the complement of 'role'. The reading corresponds the situation when I like my role which is about a little baby.

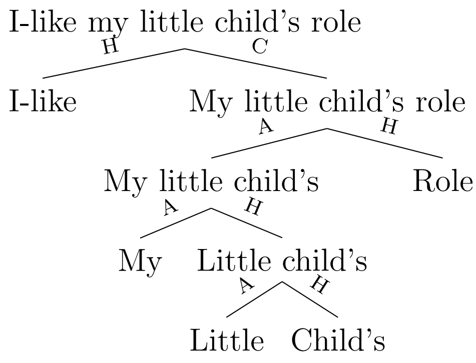


Figure 3.28: The second reading – I like the role of my little child

The second reading represents the case when I have a child and I like his (some) role. In this reading, 'role' does not find its complement.

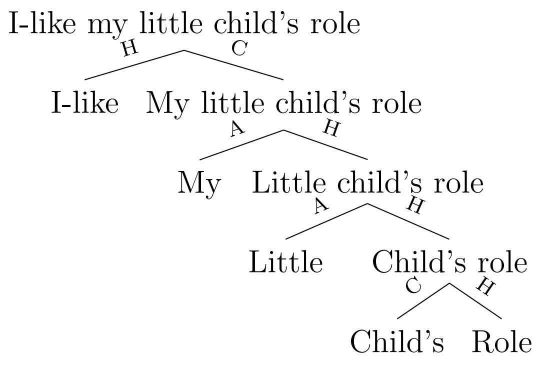


Figure 3.29: The third reading – I like my little (short) role of a child

In the last reading, 'role' is complemented by 'child' and the reading says that I like my little (short) role which is about a child.

Note that the syntactic structure where 'role' is complemented by 'my little child' is blocked as the complement can not be possessed by pronoun as it makes the complement concrete (i.e., constant) and the relation between the head and the non head constituent becomes relational (i.e., adjunction by possessives).

Also the structure where ‘role’ is possessed by ‘my’ and ‘child’ at the same time is blocked.

To draw the conclusion, the current HPSG-based grammar models the syntax of the simple declarative sentences consisting of the verb and its complement noun phrases – the head possibly complemented and possibly modified by adjectives, quantifiers or possessives. Its implementation parser syntactically analyzes the sentences modeled by the grammar theory.

4. Towards a realistic grammar – modeling the morphology of Georgian

In the previous section, we modeled the syntax of simple declarative sentences. The formal grammar (and its implemented version) can parse (provide syntactic analysis of) the sentences based on the lexicon. In this chapter, we build the set of lexical rules in the grammar. The rules make it easy to extend the lexicon by adding new lexical entries. More precisely, using these lexical rules the grammar will be able to "understand" some set of inflected forms of the canonical word form, if this canonical word form is added to the lexicon. This kind of grammar is called realistic as it roughly resembles the mental grammar as it is assumed to exist in human minds – it is sufficient to "learn" a single new word form; other inflected forms related to the "learned" one are "understood" without the need to learn them explicitly.

We start modeling the morphology of Georgian by describing lexical rules for nominals (nouns, adjectives and quantifiers). We also consider nominalization rules for adjectives and quantifiers. At the end of modeling, we implement these rules and thus make the implemented grammar more expressive.

It would not be justified if we build the formal grammar for the core part of Georgian omitting lexical rules for the verb. The study of the verb is the most difficult part of Georgian linguistics as there is very rich information encoded in the verb, which is highly inflectional. While designing lexical rules for the verb, our goal is to model a wide range of verbs and make the rules highly productive. We classify the verbs into three classes (conjugation paradigms), providing a set of lexical rules for each class The lexical rules are implemented and added to the grammar.

This chapter represents a contribution of the thesis to the formal analysis and modeling of some fundamental parts of the complex morphology of Georgian.

4.1 Lexical rules for nominals

We start with modeling and implementing lexical rules for nouns and adjuncts (adjectives and quantifiers) – rules describing the inflected forms corresponding to the marking by logical cases. In addition to these core rules we also treat other lexical rules, such as the pluralization rule producing the plural versions of nouns, the nominalization rules "converting" the adjective and quantifier into noun, and the possessive rule producing word forms marked by the possessive case.

4.1.1 Logical declension of the noun

In general, we are using fifteen inflectional forms (fourteen of them correspond to the forms marked by the logical cases and one form by the possessive case) of the the noun lemma in the lexicon of the formal grammar.[1] We design lexical rules to produce these inflected word forms from the single lexeme in the grammar. Note that we call the initial (i.e., canonical) lexical entry a *lexeme* as it does not represent a particular word form but the compact description of the lexeme – a set of the inflections.

The lexical rules are based on four declension paradigms which we defined after some manual analysis of the logical declensions of some nouns. We provide these declension paradigms in Table 4.1.

Arg. log. case	Paradigm I	Paradigm A	Paradigm E	Paradigm OU
NOM	N_1-i	N_1	N_1	N_1
ERG	N_1-ma	N_1-m	N_1-m	N_1-m
DAT	N_1-s	N_1-s	N_1-s	N_1-s
INE	N_1-ši	N_1-ši	N_1-ši	N_1-ši
LOC	N_1-ze	N_1-ze	N_1-ze	N_1-ze
COM	N_1-tan	N_1-stan	N_1-stan	N_1-stan
ADV	N_2-ad	N_2-ad	N_1-d	N_1-d
TER	N_2-amde	N_2-amde	N_1-mde	N_1-mde
INS	N_2-it	N_2-it	N_2-it	N_1-ti
GEN	N_2-is	N_2-is	N_2-is	N_1-si
ORI	N_2-isk̦en	N_2-isk̦en	N_2-isk̦en	N_1-sk̦en
ABL1	N_2-idan	N_2-idan	N_2-idan	N_1-dan
ABL2	N_2-isgan	N_2-isgan	N_2-isgan	N_1-sgan
BEN	N_2-istvis	N_2-istvis	N_2-istvis	N_1-stvis

Table 4.1: Four declension paradigms for nouns

The symbols N_1 and N_2 denote the full and non-full stems of the noun. Under the non-full stem we mean the stem with the syncope or apocope. The declension paradigms are named according to the nominative word forms of the nouns of the paradigm. For example, most nominative word forms in Paradigm OU[2] end in *o* or *u*. The given declension paradigms represent a refined version of those

[1]If we take into account the pluralized forms too, the number of word forms doubles.

[2]For example the words of foreign origins also fall into this paradigm without ending on *o* or *u*, e.g. *čai* 'tea' or *ṭramvai* 'tram'.

used in (Abzianidze, 2008) and (Chikvinidze, 2010).[3] It is not always possible to detect the syncope and apocope in the stem, that is why we require to specify both full and non-full stems in our lexical rules. After specifying both stems and the declension paradigm, the production of the noun inflections is trivial. Note that in this way the declension paradigms cover a wide range of nouns (including proper nouns and nouns of foreign origin). Though these paradigms are hand-crafted, we do not expect many exceptions – manual adding of exceptions to the lexicon would not be a problem.[4]

In Sections 4.1.3 and 4.1.3, we will show that this declension paradigms cover the pluralized nouns and the nominalized adjectives and quantifiers too.

4.1.2 Logical declension of the adjunct

Modeling the logical declension for adjectives and quantifiers is done similarly as in (Abzianidze, 2008) and (Chikvinidze, 2010). Namely, there are two declension paradigms. Adjectives and quantifiers ending with *-i* fall into the first paradigm, hence changing their word forms according to the adjunct logical cases. Other adjectives and quantifiers having the same (i.e., constant) word form in all adjunct logical cases fall into the second paradigm. The paradigms are given in Table 4.2.

Adj. log. case	Non-constant Paradigm		Constant Paradigm	
NOM	A-i	Q-i		
ERG	A-ma	Q-ma		
DAT	A-s	Q-s		
CNST			A	Q

Table 4.2: Two declension paradigms for adjectives and quantifiers

In Table 4.2 symbols A and Q denote the stem of the adjective and quantifier, respectively. As you can see, the adjective and the quantifier have the same morphological paradigms because their word form changes in a similar way (but their syntactic functions are different).

In order to determine into which paradigms the certain adjective or quantifier belongs, it is sufficient to check the last letter of its nominative word form. The adjunct ending with *i* in its nominative form falls into the Non-constant Paradigm, otherwise into the Constant Paradigm. Note that the adjunct in Constant paradigm has a unique word form marked by the CNST logical case.

4.1.3 Pluralization and possession rules

In sections 4.1.1 we discussed the logical declension paradigms for the noun. We finalize the analysis of the noun lexical rules by modeling the pluralization and the possession rules.

[3]The systems try to guess the paradigm according by the last character of the full stem and then produce the word forms. In case of a wrong guess the user has to manually edit the produced word forms. During our research we found that the probability of wrong guessing is quite high, especially in the case of nominalized adjective and quantifiers.

[4]Of course, an empirical check of this anticipation is necessary and requires additional resources.

There are two different ways how the pluralized stem (stem including the plural morpheme) is related to the “singular” stems (the full and non-full stems) of the noun. For nouns of the declension paradigms I and A, the pluralized stem is reproduced by suffixing the non-full stem (i.e., N_2) with the plural morpheme *-eb* (i.e., $N^{pl} = N_2$-eb). In the case of nouns of the paradigms E and OU, the pluralized stem represents $N^{pl} = N_1$-eb – the full stem affixed with the plural morpheme. In Table 4.3, the second and third columns are the same declension paradigms (the difference is only in stems which are not part of the paradigm).

Arg. log. case	Paradigm I & A	Paradigm E & OU	Paradign I
NOM	N_2-eb-i	N_1-eb-i	N_1-i
ERG	N_2-eb-ma	N_1-eb-ma	N_1-ma
DAT	N_2-eb-s	N_1-eb-s	N_1-s
INE	N_2-eb-ši	N_1-eb-ši	N_1-ši
LOC	N_2-eb-ze	N_1-eb-ze	N_1-ze
COM	N_2-eb-tan	N_1-eb-tan	N_1-tan
ADV	N_2-eb-ad	N_1-eb-ad	N_2-ad
TER	N_2-eb-amde	N_1-eb-amde	N_2-amde
INS	N_2-eb-it	N_1-eb-it	N_2-it
GEN	N_2-eb-is	N_1-eb-is	N_2-is
ORI	N_2-eb-isḳen	N_1-eb-isḳen	N_2-isḳen
ABL1	N_2-eb-idan	N_1-eb-idan	N_2-idan
ABL2	N_2-eb-isgan	N_1-eb-isgan	N_2-isgan
BEN	N_2-eb-istvis	N_1-eb-istvis	N_2-istvis

Table 4.3: Pluralization and declension paradigms for nouns

Any pluralized noun is declined according to paradigm I. Namely, based on the token-identical pluralized stems N_1^{pl} and N_2^{pl} (where $N_1^{pl} = N_2^{pl} = N^{pl}$), the declension of the plural word forms follows the declension paradigm I. For a demonstration of the identity between the pluralized declension paradigm and the paradigm I, we give both in Table 4.3.

For producing the possessive form of the noun the same information – two stems and the declension paradigm, is sufficient as in the case of pluralization and declension.

Case	Paradigm I	Paradigm A	Paradigm E	Paradigm OU
GEN	N_2-is	N_2-is	N_2-is	N_1-si
POSS	N_2-is	N_2-is	N_2-is	N_1-s

Table 4.4: Inflection for the possessive case

Table 4.4 shows how the possessive form is produced from the stems depending on the declension paradigms. Not that the difference between the possessive and genitive word forms occurs only for nouns of the paradigm OU.

4.1.4 Nominalization of adjectives and quantifiers

In Georgian, the nominalization of adjectives is quite common. This means that the adjective gains the (morphological and syntactic) characteristics of the noun (gets nominalized) and is used as a noun referring to the set of entities having the property expressed by the adjective.

For example, the sentence (1) uses the nominalized adjective 'little' referring to the entities which are little. The entities are further specified by the context and they represent animate entities which are little. The most common way of understanding the sentence is given in the English translation of (1).

(1)	*patar-eb-i*	*tamaš-ob-en*
	Little-PL-NOM	play-PRS-IND-IPFV-3PL
	Little babies/children are playing	

In the case of the nominalization of the quantifier, the domain over which the variable is quantified is restricted by the context. Since the nominalized adjectives and quantifiers syntactically behave as nouns we are going to introduce the nominalization of adjective in our formal grammar.[5]

The analysis showed that nominalized adjectives and quantifiers are declined according to one of the declension paradigms we have tailored for the noun (Table 4.1). Hence, in order to decline the nominalized adjectives and quantifiers we need to encode in their lexemes the information about the declension paradigm and two stems, like for noun lexemes. We model the nominalization with the a separate lexical rule, which produces the noun lexeme from the adjective and the quantifier lexemes.

The details about the nominalization are given in sections 4.1.5 and 4.1.6.

4.1.5 Putting all together in the formal grammar

After setting up the paradigms and specifying inflectional rules working on each paradigms, we are ready to model all this collected knowledge in the HPSG framework.

First of all we start by updating the type hierarchy of the grammar – adding new types corresponding to lexemes (Fig.4.1). In order to distinguish words and phrases from the lexemes, we introduce two new subtypes of *sign*: *expr* – the type for linguistic expressions such as words and phrases, and *lexeme* – the type for abstract linguistic objects such as lexemes. Note that the types (of the values) of the HEAD_DTR and NONH_DTR features are not *sign* but *expr*. For the horizontal lexical generalization, we introduce the feature LEX of the type *lexeme* at *word*.[6]

One of the subtypes of *lexeme* is *nominal_lex*, corresponding to nominal lexemes. We classify the nominal lexemes as the noun (*noun_lex*), quantifier (*qnt_lex*) and adjective (*adj_lex*) lexemes. The DECL feature encodes the information about the declension paradigm. The feature is appropriate to all nominal lexemes as we are modeling the nominalization rules too. For the lexical rules from lexemes to

[5]If we were also modeling the semantics, then modeling the nominalization would be a much more complex task.

[6]Note that this feature is also useful for implementing the grammar in TRALE.

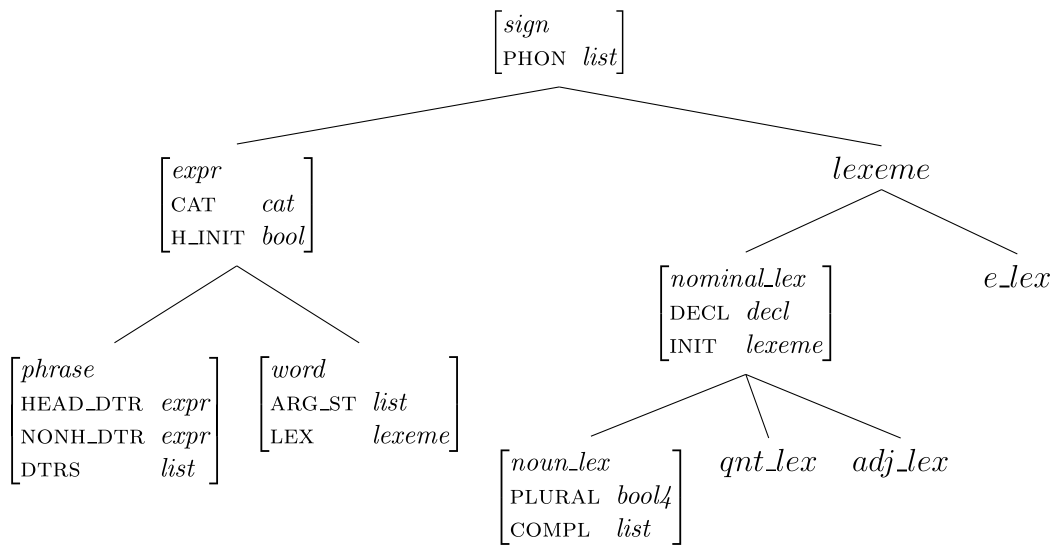


Figure 4.1: The revised *sign* type with its subtypes

lexemes, we use the feature INIT to keep track the origin lexeme.[7] If the lexeme has no ancestor, then the value of the INIT feature is *e_lex* (the empty lexeme).

The noun lexeme, in addition to the features specifying its declension paradigm and initial lexeme, encodes the information (in the feature PLURAL) on whether pluralization is valid for word forms of the lexeme. This information is necessary to avoid ungrammatical lexical entries like e.g. *haer-eb-i* 'air-PL-NOM'.

During the composition of several lexical rules, the "two-valued" *bool* type is not enough to express the pluralization information, hence we extend the type to *bool4*, which is used in further versions of the implemented grammar. The information about the complement is now encoded on the lexeme level (by the feature COMPL), as it is common for all word forms of the lexeme.

In the grammar we introduce lexical rules for reproducing the lexical entries (i.e., words) and lexemes too. This kind of organization of lexical rules is quite compact and natural at the same time. The schemata in Fig. 4.2 express the lexical rules with their domain and range. Lexemes are placed at the top and words forms at the bottom of the schemata (Fig. 4.2). The number of lexical rules is five and they are represented as transitions (i.e., rewriting rules) between the descriptions. For clarity, domains and ranges of some rules are partitioned, hence those rules are repeated in the schemata.

We briefly describe the lexical rules. The *adjunct_decl* rule applies to quantifier and adjective lexemes and produces word forms of quantifiers and adjectives, respectively, marked by all possible adjunct logical cases. Phonological morphing is done according to Table 4.2.

The *nominalization* rule produces nominalized (i.e., noun) lexemes from quantifier and adjective lexemes. Note that the feature PLURAL of the noun lexeme, which is produced from the quantifier lexeme, has the value *minus*, in order to avoid pluralization of the nominalized quantifiers.

The pluralized noun lexeme is generated by the *pluralization* rule. Note that the rule applies to both noun lexemes and nominalized adjective lexemes (both

[7]This feature is also very helpful during the implementation of the lexical rules.

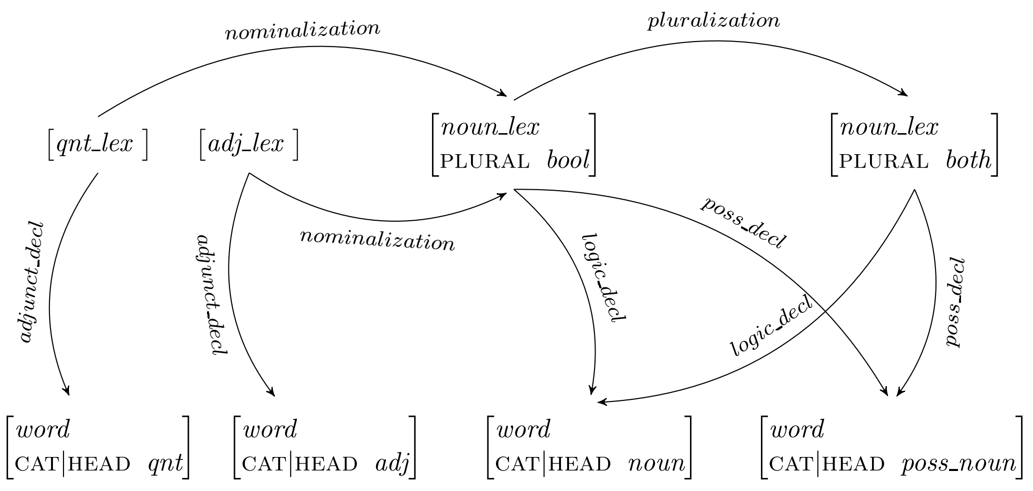


Figure 4.2: Schemata of the lexical rules on nominals

denoted by the *noun_lex* type).

The *logic_decl* and *poss_decl* lexical rules produce word forms marked by the argument logical cases and possessive case, respectively. The construction of the phonology is done according to Table 4.1 and Table 4.4. The *logic_decl* rule is the most productive rule, generating fourteen different word forms from a lexeme.

We do not explore the lexical rules further in the HPSG framework, but implement them in the next section. Further details about each lexical rules will be revealed in its implementation as TRALE grammar is faithful to its "hand-written" grammar in the HPSG framework.

4.1.6 Implementation

We start with implementing the lexical rules presented in the previous section. While implementing the rules some technical issues will be mentioned along with their solutions. At the end we demonstrate how the lexical rules work inside the grammar.

The main issue while implementing the lexical rules is how to encode two stems in the structure, or in general, how to make several features encode the different phonology information and at the same time enable the string operations (those available for the value of the PHON feature) on them. The problem is that PHON has a special status in lexical rules – its value is automatically carried over to morphing part of the rule where powerful matching functionalities are applicable to it. Another related problem is that only `a_/1` atoms are accessible in the morphing section (and of course the value of PHON). As the string matching and concatenation operations are not available on `a_/1` atoms in TRALE, we encode both stems in the PHON feature, separated by the underscore. With this solutions, we are able to use TRALE's string matching and concatenating operations on both stems.

The most important rule in the mentioned lexical rules is the *logic_decl* rule and this is why we discuss it here. We build the rule in the way that it can generates fourteen different word forms for a lexeme. The part of implementation of the *logoc_decl* lexical rule is given in List. 4.1 (we skipped the morphing part due to its length).

In the rule, the lexeme and its complements are copied into the produced word. The produced lexical entry is unspecified. The case and the number categories are evaluated in the condition of the rule. Here, we use the power of Prolog and TRALE clauses. Namely, in order to make the rule generate fourteen different word form, we "loop" over the logical case types by the predicate `isLogCase/2`, defined in terms of fourteen clauses. The predicate represents the set of fourteen pairs of argument logical case type and the corresponding Prolog `a_/1` atom. The second argument of `isLogCase/2` is used to carry over the information about the case type to the morphs part of the lexical rule (namely, in the `when`'s Prolog goal). We use the predicate `isParadigm/2` for the same purpose – to carry over the information about the declension paradigm to the Prolog goal.

Listing 4.1: The lexical rule for logical declension

```
logic_decl  ##
     (L, noun_lex,
          compl:Compl,
          decl:Par,
          plural:Pl).
**>
     (word,
          cat:(cat,
               head:(noun,
                    case:Case,
                    num:Num),
               val:(spec:unspec,
                    comp:Comp)),
          arg_st:Comp,
          lex:L).

if   isLogCase(Case, a_ C),
     isParadigm(Par, a_ P),
     (Pl=(plus;minus) -> Num=sing; Num=plur)

morphs
% Full version of the rule
% is available in appendix D
...
```

After the information about where the declension paradigm and case are kept in Prolog's variables, we can easily generate all possible word forms with the help of TRALE's phonology matching and Prolog's goals (For the details see appendix D).

While implementing lexical rules we have to change the lexical rule depth default value 2 by 3. The reason is that in our lexical rules there is a case where a sequence of three lexical rules is applied to a lexeme. Namely, the word form for the nominalized adjective in plural marked by the ergative case is generated by applying the *nominalization*, *pluralization* and *logic_decl* rules to the adjective lexeme (Fig. 4.2).

To show how the lexical rules work as a system, we consider the example of the word form which is generated by three different lexical rules at the same time. The AVM of the description of the word form – the nominalized adjective in plural, marked by the ergative case, is given in Fig 4.3. Note that the word form is one of 33 word forms[8] generated from the adjective lexeme (its corresponding TFS is *adj_lex* in Fig. 4.3).

[Little-PL-ERG]

$$\begin{bmatrix} \textit{word} & \\ \text{PHON} & \langle \textit{patarebma} \rangle \\ \text{CAT} & \begin{bmatrix} \textit{cat} & \\ \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \textit{erg} \\ \text{NUM} & \textit{plur} \\ \text{PERS} & \textit{per3} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \textit{comps} & \\ \text{COMP} & \boxed{0} \\ \text{SPEC} & \textit{unspec} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \boxed{0} \\ \text{LEX} & \begin{bmatrix} \textit{noun\_lex} & \\ \text{PHON} & \langle \textit{patareb\_patareb} \rangle \\ \text{PLURAL} & \textit{both} \\ \text{DECL} & \textit{decl\_i} \\ \text{COMPL} & \boxed{0} \\ \text{INIT} & \begin{bmatrix} \textit{noun\_lex} & \\ \text{PHON} & \langle \textit{patara\_patar} \rangle \\ \text{PLURAL} & \textit{plus} \\ \text{DECL} & \boxed{1} \\ \text{COMPL} & \boxed{0}\,\langle\rangle \\ \text{INIT} & \begin{bmatrix} \textit{adj\_lex} & \\ \text{PHON} & \langle \textit{patara\_patar} \rangle \\ \text{DECL} & \boxed{1}\ \textit{decl\_a} \\ \text{INIT} & \begin{bmatrix} \textit{e\_lex} & \\ \text{PHON} & \textit{list} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 4.3: A lexical entry produced by three lexical rules

The values of LEX and *init* features show us in which order the lexical rules were applied on the initial lexeme, or the history of the generation, and how both stems encoded as a unique phonology are changing at the same time in different ways. This kind of information is useful for capturing the horizontal generalizations and at the same time it represents data valuable for implementing and maintaining the grammar.

[8]Namely, 3 of them are adjective forms and 30 forms are nominalized forms in plural and singular, marked by logical cases and the possessive case.

4.2 Lexical rules for the verbs

In this section, we discuss the verb and its conjugation system. The traditionally assumed approach to the verb conjugation is critically assessed and an approach better suited to Georgian is presented. Based on an analysis of a wide range of verbs, we propose three conjugation paradigms (i.e., verb classes, as opposed to the traditional system of four classes). Verbs of the same paradigm do not have the same conjugation system only, but also the same case alignment of explicit arguments. At the end, we implement the conjugation system of each paradigm.

4.2.1 The verb and its conjugation

We will discuss the verb and the traditionally assumed approach to its conjugation, show that the approach is not natural for the verb with the polypersonal property. We will also talk about the traditional verb classification, which has many of exceptions.

In the traditional grammar, the verb is conjugated according to its subject. This tradition is carried over from the Indo-European languages where mostly only the subject agrees with the verb, but as we have seen in the previous chapters, the verb can agree with other arguments in several persons. The result of this subject-centered approach is that often the polypersonal property of the verb is ignored and often most word forms are not considered.[9] We will call this kind of conjugation *unary conjugation* in contrast to *polypersonal conjugation* – a conjugation according to several arguments of the verbal predicate.

To continue with the topic of subject, it appears that there is no rule in Georgian according to which one can say which constituent is the subject. Sometimes the subject can be marked by the nominative, ergative or dative cases. Moreover, the subject and the object are represented by the same morpheme in the verb. For example in (2), the subject is expressed by *m* morpheme and the same morpheme is used to express the object of the verb "dance" in (3). Note that the subject is not expressed by any morpheme in (3).[10]

(2) *m-i-qvar-xar*
1SING-love-PRS-IND-IPFV-2SING
I love you

(3) *m-e-cek̦v-eb-i*
1SING-dance-PRS-IND-IPFV-2SING
You are dancing with me

That is why the traditional Georgian grammar uses the notions of morphological subject and morphological object. In (2), the morphological and syntactic subjects are the same but in (3), they are different.

In the traditional grammar, the classification of verbs is done according to *screeves* (Table 1.2). For example, verbs which can produce word forms for the same set of screeves are considered the same class. Based on this classification

[9]For example, the word form agreeing with the first person and the second person is ignored, since according to the standard conjugation there is at least one third person involved.

[10]This kind of examples are also emphasized in (Pkhakadze, 2004) along with criticism of the alleged verbal conjugation.

there are four verb classes. This kind of classification also attempts to characterize verb classes according to the transitivity feature but it fails, since there are quite a few exceptions for some classes.[11] This kind of classification is not suited for the conjugation paradigms.[12] Note that this classification in four classes is used in (Meurer, 2009).[13]

In the next sections, we are going to analyze the verbal conjugation without using any undefined notions (such as subject and object). Then we classify verbs into three conjugation paradigms covering a wide range of verbs.

4.2.2 The verb conjugation paradigms

We organize the section in three subsections. Namely, in each subsection we give the conjugation paradigms represented as a table. The paradigms are built on the basis of the analysis of verbs and their polypersonal conjugation (as opposed to unary conjugation). In this thesis, we are dealing only with the core tenses, namely, expressed by the present, imperfect, future and aorist screeves (Table 1.2). These screeves roughly correspond to the English present indefinite (simple), past progressive (continuous), future indefinite and past indefinite (simple) tenses, respectively.[14]

The conjugation paradigm 1

The conjugation systems of the paradigm 1 are given in Table 4.5 and Table 4.6. The table encodes the binary conjugation system.

Persons	Present simple		Past progressive	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-v-[ST]-(var)	[P]-v-[ST]-(var)-t	[P]-v-[ST]-i	[P]-v-[ST]-i-t
	[P]-[ST]-(xar)	[P]-[ST]-(xar)-t	[P]-[ST]-i	[P]-[ST]-i-t
	[P]-[ST]-s	[P]-[ST]-en/an	[P]-[ST]-a	[P]-[ST]-nen
X_2-2	[P]-g-[ST]-(var) [P]-g-[ST]-(var)-t	[P]-g-[ST]-(var)-t	[P]-g-[ST]-i [P]-g-[ST]-i-t	[P]-g-[ST]-i-t
	[P]-g-[ST]-s [P]-g-[ST]-t	[P]-g-[ST]-en/an	[P]-g-[ST]-a [P]-g-[ST]-a-t	[P]-g-[ST]-nen
X_1-1	[P]-m-[ST]-(xar) [P]-gv-[ST]-(xar)	[P]-m-[ST]-(xar)-t [P]-gv-[ST]-(xar)-t	[P]-m-[ST]-i [P]-gv-[ST]-i	[P]-m-[ST]-i-t [P]-gv-[ST]-i-t
	[P]-m-[ST]-s [P]-gv-[ST]-s	[P]-m-[ST]-en/an [P]-gv-[ST]-en/an	[P]-m-[ST]-a [P]-gv-[ST]-a	[P]-m-[ST]-nen [P]-gv-[ST]-nen
	1-NOM 2-DAT 3-DAT		1-NOM 2-DAT 3-DAT	

Table 4.5: Part 1 of the conjugation paradigm 1

[11]In spite of the fact, the classes still use names as *transitive class* or *intransitive class*.

[12]It is obvious that the classification, which does not consider the (even wrong) conjugation at all, cannot be used for the verb conjugation.

[13]Moreover, the author claims that the correct verbal frames are not easily deducible from this classification.

[14]Note that these tenses are already introduces in the current version of the formal grammar.

Each word pattern represents the sequence of a preverb ([P]), followed by a morpheme insert (possibly empty one), a stem ([ST], which is characteristic for the tense – a stem is shared by the word forms of the same tense) and word-final morpheme(s). A morpheme in parenthesis is only used by some verbs. A string morpheme$_1$/morpheme$_2$ means that, depending on the verb, exactly one of the morphemes is used in the conjugation.

Persons	Past simple		Future indefinite	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-v-[ST]-e/i	[P]-v-[ST]-e/i-t	[P]-v-[ST]-i	[P]-v-[ST]-t
	[P]-[ST]-e/i	[P]-[ST]-e/i-t	[P]-[ST]	[P]-[ST]-t
	[P]-[ST]-a/o	[P]-[ST]-es	[P]-[ST]-s	[P]-[ST]-en/an
X_2-2	[P]-g-[ST]-e/i [P]-g-[ST]-e/i-t	[P]-g-[ST]-e/i-t	[P]-g-[ST] [P]-g-[ST]-t	[P]-g-[ST]-t
	[P]-g-[ST]-o/a [P]-g-[ST]-o/a-t	[P]-g-[ST]-es	[P]-g-[ST]-s [P]-g-[ST]-t	[P]-g-[ST]-en/an
X_1-1	[P]-m-[ST]-e/i [P]-gv-[ST]-e/i	[P]-m-[ST]-e/i-t [P]-gv-[ST]-e/i-t	[P]-m-[ST] [P]-gv-[ST]	[P]-m-[ST]-t [P]-gv-[ST]-t
	[P]-m-[ST]-a/o [P]-gv-[ST]-a/o	[P]-m-[ST]-es [P]-gv-[ST]-es	[P]-m-[ST]-a [P]-gv-[ST]-s	[P]-m-[ST]-en/an [P]-gv-[ST]-en/an
	1-ERG 2-DAT 3-NOM		1-NOM 2-DAT 3-DAT	

Table 4.6: Part 2 of the conjugation paradigm 1

The symbol X_i denotes a variable over persons, except for the i^{th} person. Therefore, X_2-2 means two combinations ⟨1, 2⟩ ⟨3, 2⟩. If the same word form is used to express both the 2nd argument in singular and in plural, then there is just one word pattern in the table cell.

At the bottom of the table, the case structure is specified. The number expresses the argument position. With this ordering and case assignments we determine the order of the arguments for every verb of the paradigm. Note that each argument is distinguishable from other arguments on the basis of their change according to tenses. Note that some verbs of the paradigms are lacking the 2nd and/or 3rd arguments, but the first arguments is always presented. We call it by *semantic subject.* We call it semantic subject since it is the main participant of the event expressed by the verb (e.i., it is the agent or the only participant of the event).

The unary conjugation (i.e., the conjugation according to the unique explicit argument) is expressed in the sub table of the binary conjugation. Namely, the subtable in the row of "X-3".

The conjugation paradigm 2

The conjugation paradigm 2 is exemplified in Table 4.8 and Table 4.8. The verbs from this paradigm can also have the polypersonal conjugation. The maximum number of explicit arguments of the paradigm is two. The first and second arguments are marked constantly by the nominative and dative case, respectively. Therefore the semantic subject is always marked by the nominative case.

Persons	Present simple		Past progressive	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-v-[ST]-i	[P]-v-[ST]-i-t	[P]-v-[ST]-i	[P]-v-[ST]-i-t
	[P]-[ST]-i	[P]-[ST]-i-t	[P]-[ST]-i	[P]-[ST]-i-t
	[P]-[ST]-a	[P]-[ST]-ian	[P]-[ST]-a	[P]-[ST]-nen
X$_2$-2	[P]-g-[ST]-i	[P]-g-[ST]-i-t	[P]-g-[ST]-i	[P]-g-[ST]-i-t
	[P]-g-[ST]-i-t		[P]-g-[ST]-i-t	
	[P]-g-[ST]-a	[P]-g-[ST]-ina	[P]-g-[ST]-a	[P]-g-[ST]-nen
	[P]-g-[ST]-a-t		[P]-g-[ST]-a-t	
X$_1$-1	[P]-m-[ST]-i	[P]-m-[ST]-i-t	[P]-m-[ST]-i	[P]-m-[ST]-i-t
	[P]-gv-[ST]-i	[P]-gv-[ST]-i-t	[P]-gv-[ST]-i	[P]-gv-[ST]-i-t
	[P]-m-[ST]-a	[P]-m-[ST]-ian	[P]-m-[ST]-a	[P]-m-[ST]-nen
	[P]-gv-[ST]-a	[P]-gv-[ST]-ian	[P]-gv-[ST]-a	[P]-gv-[ST]-nen
	1-NOM 2-DAT		1-NOM 2-DAT	

Table 4.7: Part 1 of the conjugation paradigm 2

Note that for the conjugation of the verb in this paradigm, one has to explicitly encode the information about the suffix for the past tense. There are only four options for the set of suffixes: [e, e, a], [e, e, o], [i, i, a] and [i, i, o]. By knowing all four stems, a set of suffixes for the past tense and applicable preverbs for each tense, we can generate 84 word forms.

Persons	Past simple		Future indefinite	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-v-[ST]-e/i	[P]-v-[ST]-e/i-t	[P]-v-[ST]-i	[P]-v-[ST]-i-t
	[P]-[ST]-e/i	[P]-[ST]-e/i-t	[P]-[ST]-i	[P]-[ST]-i-t
	[P]-[ST]-a/o	[P]-[ST]-nen	[P]-[ST]-a	[P]-[ST]-ian
X$_2$-2	[P]-g-[ST]-e/i	[P]-g-[ST]-e/i-t	[P]-g-[ST]-i	[P]-g-[ST]-i-t
	[P]-g-[ST]-e/i-t		[P]-g-[ST]-i-t	
	[P]-g-[ST]-a/o	[P]-g-[ST]-nen	[P]-g-[ST]-a	[P]-g-[ST]-ian
	[P]-g-[ST]-a/o-t		[P]-g-[ST]-a-t	
X$_1$-1	[P]-m-[ST]-e/i	[P]-m-[ST]-e/i-t	[P]-m-[ST]-i	[P]-m-[ST]-i-t
	[P]-gv-[ST]-e/i	[P]-gv-[ST]-e/i-t	[P]-gv-[ST]-i	[P]-gv-[ST]-i-t
	[P]-m-[ST]-a/o	[P]-m-[ST]-nen	[P]-m-[ST]-a	[P]-m-[ST]-ian
	[P]-gv-[ST]-a/o	[P]-gv-[ST]-nen	[P]-gv-[ST]-a	[P]-gv-[ST]-ian
	1-NOM 2-DAT		1-NOM 2-DAT	

Table 4.8: Part 2 of the conjugation paradigm 2

The conjugation paradigm 3

The conjugation paradigm is given in two Tables 4.9 and Table 4.10. The paradigm has at most two explicit arguments. The semantic subject is always in the dative case and the second argument in the nominative case.

In the word patterns, some last letters are deleted from the stem, while new morpheme is appended on it. Moreover, there are also other phonological processes going in the front part of the stem. For example, the stems starting with the plosive and ejective sounds are prefixed by the fricative sounds. Further details about the phonological changes in the word forms can be found in the source code of the conjugation lexical rules.

Persons	Present simple		Past progressive	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-m-[ST]	[P]-gv-[ST]	[P]-m-[ST]	[P]-gv-[ST]
	[P]-g-[ST]	[P]-g-[ST]-~~s~~-t	[P]-g-[ST]	[P]-g-[ST]-t
	[P]-[ST]	[P]-[ST]-~~s~~-t	[P]-[ST]	[P]-[ST]-t
X_2-2	[P]-m-[ST]-~~s/a~~-xar/i [P]-m-[ST]-~~s/a~~-xar/i-t	[P]-gv-[ST]-~~s/a~~-xar/i [P]-gv-[ST]-~~s/a~~-xar/i-t	[P]-m-[ST]-~~a~~-i [P]-m-[ST]-~~a~~-i-t	[P]-gv-[ST]-~~a~~-i [P]-gv-[ST]-~~a~~-i-t
	[P]-[ST]-~~s/a~~-xar/i [P]-[ST]-~~s/a~~-xar/i-t	[P]-[ST]-~~s/a~~-xar/i [P]-[ST]-~~s/a~~-xar/i-t	[P]-[ST]-~~a~~-i [P]-[ST]-~~a~~-i-t	[P]-[ST]-~~a~~-i [P]-[ST]-~~a~~-i-t
X_1-1	[P]-g-[ST]-~~s/a~~-var/i [P]-g-[ST]-~~s/a~~-var/i-t	[P]-g-[ST]-~~s/a~~-var/i-t	[P]-g-[ST]-~~a~~-i [P]-g-[ST]-~~a~~-i-t	[P]-g-[ST]-~~a~~-i-t
	[P]-v-[ST]-~~s/a~~-var/i [P]-v-[ST]-~~s/a~~-var/i-t	[P]-v-[ST]-~~s/a~~-var/i [P]-v-[ST]-~~s/a~~-var/i-t	[P]-v-[ST]-~~a~~-i [P]-v-[ST]-~~a~~-i-t	[P]-v-[ST]-~~a~~-i [P]-v-[ST]-~~a~~-i-t
	1-DAT 2-NOM		1-DAT 2-NOM	

Table 4.9: Part 1 of the conjugation paradigm 3

Persons	Past simple		Future indefinite	
	SING-NUM	PL-NUM	SING-NUM	PL-NUM
X-3	[P]-m-[ST]	[P]-gv-[ST]	[P]-m-[ST]	[P]-gv-[ST]
	[P]-g-[ST]	[P]-g-[ST]-t	[P]-g-[ST]	[P]-g-[ST]-~~s~~-t
	[P]-[ST]	[P]-[ST]-t	[P]-[ST]	[P]-[ST]-~~s~~-t
X_2-2	[P]-m-[ST]-~~a~~-e/i [P]-m-[ST]-~~a~~-e/i-t	[P]-gv-[ST]-~~a~~-e/i [P]-gv-[ST]-~~a~~-e/i-t	[P]-m-[ST]-~~a~~-i [P]-m-[ST]-~~a~~-i-t	[P]-gv-[ST]-~~a~~-i [P]-gv-[ST]-~~a~~-i-t
	[P]-[ST]-~~a~~-e/i [P]-[ST]-~~a~~-e/i-t	[P]-[ST]-~~a~~-e/i [P]-[ST]-~~a~~-e/i-t	[P]-[ST]-~~a~~-i [P]-[ST]-~~a~~-i-t	[P]-[ST]-~~a~~-i [P]-[ST]-~~a~~-i-t
X_1-1	[P]-g-[ST]-~~a~~-e/i [P]-g-[ST]-~~a~~-e/i-t	[P]-g-[ST]-~~a~~-e/i-t	[P]-g-[ST]-~~a~~-i [P]-g-[ST]-~~a~~-i-t	[P]-g-[ST]-~~a~~-i-t
	[P]-v-[ST]-~~a~~-e/i [P]-v-[ST]-~~a~~-e/i-t	[P]-v-[ST]-~~a~~-e/i [P]-v-[ST]-~~a~~-e/i-t	[P]-v-[ST]-~~a~~-i [P]-v-[ST]-~~a~~-i-t	[P]-v-[ST]-~~a~~-i [P]-v-[ST]-~~a~~-i-t
	1-DAT 2-NOM		1-DAT 2-NOM	

Table 4.10: Part 2 of the conjugation paradigm 3

To draw a conclusion, we wish to mention that most of the verbs are falling into these paradigms. Note that this is explained by the fact that we are already providing the verb with four stems (for four tenses, respectively).

Note that the conjugation paradigms we got coincide with the diatheses of (Melikishvili, 2001).[15] Note that our approach of finding these paradigms was simply guided by the morphology of personal marking, in contrast to (Melikishvili, 2001), which uses several linguistic descriptions at the same time.

We have to mention verbs which do not fall at this moment into any of the three conjugation paradigms. Most of these verbs use two different stems inside a tense.[16] The number of these verbs is not large. They need further investigation to find out how diversely they are conjugated.

4.2.3 Implementation

For the implementation of the conjugation rules, we introduce verb lexemes. Verb lexemes encode information about four stems in the PHON feature in the same way as noun lexemes encode the stems. Moreover, the verb lexeme includes information about the conjugation paradigm (i.e., class). The conjugation paradigms are described in terms of features, mainly encoding information about the sets of suffixes and the arity of conjugation. The list of applicable preverbs along with the implicit arguments (since they are capturing lexical generalizations) is also encoded in the verb lexeme.

From the technical point of view, we implement the predicates for looping over the person and number categories, also over the list of preverbs. To avoid complications in the conjugation rules we introduce Prolog's definite clauses to model the predicates used to treat the valid phonological changes in word forms.

We do not present the conjugation rules in this thesis for space reasons. Further details about the lexical rules can be found on the CD attached to the hard copy of the thesis, or at the following address: `https://sites.google.com/site/lashabzianidze/thesis`. Also, further updates of the formal and implemented grammar will be available at the address.

[15]After defining the diatheses, (Melikishvili, 2001) gives further classification of verbs inside the diatheses.

[16]These verbs also does not fall in the scope of diatheses of (Melikishvili, 2001).

References

Abzianidze, L. 2008. *Georgian Simple Sentence Syntactic Spellchecker based on Mathematical Methods (Experimental version).* Master thesis, Tbilisi State University. in Georgian, http://gllc.ge/publications/articles/L.Abzianidze_BT.pdf.

Butskhrikidze, M. 2002. *The Consonant Phonotactics of Georgian.* Netherlands Graduate School of Linguistics.

Carpenter, B. 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

Carpenter, B. 1993. Skeptical and credulous default unification with applications to templates and inheritance. *In:* E. J. Briscoe, A. Copestake, & de Paiva, V. (eds), *Defaults, Inheritance and the Lexicon.* Cambridge University Press.

Chikvinidze, M. 2010. *1st version of the Georgian-English-German Two-Way Translator Computer System Constructed by Mathematical Methods.* Master thesis, Tbilisi State University, Georgia.

Flickinger, D. 1987. *Lexical Rules in the Hierarchical Lexicon.* Phil. Dissertation, Stanford University.

Hillery, P.J. 2006. *The Georgian Language – An outline grammatical summary.* http://www.armazi.com/georgian/.

Levine, R. D., & Meurers, D. W. 2006. Head-Driven Phrase Structure Grammar: Linguistic Approach, Formal Foundations, and Computational Realization. *In:* Brown, Keith (ed), *Encyclopedia of Language and Linguistics. Second Edition.* Oxford: Elsevier.

Melikishvili, D. 2001. *Conjugation system of Georgian verb.* Logos press.

Melnik, N. 2007. From "Hand-Written" to Computationally Implemented HPSG Theories. *Research on Language and Computation*, **5**(2), 199–236.

Meurer, P. 2009. Logic, Language, and Computation. Springer-Verlag.

Meurers, W. D., Penn, G., & Richter, F. 2002. A Web-based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing. *Pages 18–25 of: Effective Tools and Methodologies for Teaching NLP and CL.* ACL.

Moshier, M. 1988. *Extensions to unification grammar for the description of programming languages.* Ph.D. thesis, Ann Arbor, MI, USA.

Moshier, M., Rounds W. 1987. A Logic for Partially Specified Data Structures. *Pages 156–167 of: Principles of Programing Languages.*

Penn, G., & Abdolhosseini, M. H. 2003a. *The Attribute Logic Engine with TRALE extensions.* User's Guide.

Penn, G., & Abdolhosseini, M. H. 2003b. *TRALE reference manual.* draft.

Penn, G., Meurers, D., De Kuthy, K., H., Abdolhosseini M., Metcalf, V., Muller, S., & H., Wunsch. 2003. *Trale Milca Environment v. 2.5.0.* User's Manual (Draft).

Penn, G. B. 2000. *The Algebraic Structure of Attributed Type Signatures.* Phil. Dissertation, Carnegie Mellon University.

Pkhakadze, K., Abzianidze L. Maskharashvili A. 2008a. Georgian language's theses. *In: Reports vol. 34, Seminar of I. Vekua Institute of Applied Mathematics.* http://www.viam.science.tsu.ge/report/vol34/pkhakadze_2.pdf.

Pkhakadze, K., Abzianidze L. Maskharashvili A. 2009a. The mathematical analysis of Georgian declarative verbs. *In: Reports of Enlarged Session of the Seminar of I. Vekua Institute of Applied Mathematics.* http://gllc.ge/publications/articles/VIAM2009_MA_of_GDV.pdf.

Pkhakadze, K., Abzianidze L. Maskharashvili A. Bakradze N. Chichua G. Gurasashvili L. Pkhakadze N. Vakhania N. 2008b. *The 2-Stage Logical Grammar of Georgian Language And The 1-Stage Voice Managed Georgian Intellectual Computer System.* Online publication, http://gllc.ge/publications/online_issues/2Stage_LGofGL&1Stage_VMGICS.pdf.

Pkhakadze, K., Chichua G. Vashalomidze A. Abzianidze L. Maskharashvili A. Pkhakadze N. Chikvinidze M. 2009b. *Toward Complete Mathematical and Mechanical Foundation of the Georgian Language and Thinking.* Online publication, http://gllc.ge/publications/online_issues/Foundation_of_GLT.pdf.

Pkhakadze, K. 2004. *Pre-verbal Semantic Unit, Problem of Personal Mark Signs, Integral and Non-Integral Verbal Word-Semantics and Incomplete or First Semantic Classification of Georgian Verbs.* in Georgian.

Pkhakadze, K. 2005. *About Logical Declination and Lingual Relations in Georgian.* in Georgian, http://gllc.ge/publications/issues/Jurnali_1_2005.pdf.

Pollard, C. 1997. *Lectures on the foundations of HPSG.* Course Notes.

Pollard, C., & Sag, I. A. 1987. *Information-Based Syntax and Semantics. Volume 1: Fundamentals.* Stanford: CSLI Publications.

Pollard, C., & Sag, I. A. 1994. *Head-Driven Phrase Structure Grammar.* Chicago: The University of Chicago Press.

Rosen, A. 2010a. *How to write a TRALE grammar.* Course in Linguistic Theory and Grammar Formalisms, http://utkl.ff.cuni.cz/~rosen/public/trale_syntax.pdf.

Rosen, A. 2010b. *TRALE.* Course in Linguistic Theory and Grammar Formalisms, http://utkl.ff.cuni.cz/~rosen/public/trale.pdf.

Sag, I. A., Wasow, T., & Bender, E. M. 2003. *Syntactic Theory: A Formal Introduction, 2nd edition.* CSLI.

Uszkoreit, H. 1996. *A personal note on the essence of HPSG and its role in computational linguistics.* `http://www.coli.uni-sb.de/~hansu/hpsg1.html`.

Vamling, K. 1989. *Complementation in Georgian.* Ph.D. Thesis, Lund University.

A. GeoGram ver.1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% SIGNATURE %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

type_hierarchy
bot
  sign  phon:list  cat:cat  h_init:bool
    word  h_init:plus
    phrase  head_dtr:sign  nonh_dtr:sign  dtrs:list
  cat  head:head  val:list
  head
    verb
    noun  num:num  case:case  pers:per3
  case
    nom   % nominative   N-i
    ins   % instrumental N-ith
    gen   % genetive     N-is(si)
    ori   % orientative  N-isken
    abl1  % ablative1    N-idan
    abl2  % ablative2    N-isgan
    ben   % benefactive  N-isthvis
    erg   % ergative     N-ma
    ine   % inessive     N-shi
    dat   % dative       N-s
    loc   % locative     N-ze
    com   % comitative   N-than
    adv   % adverbial    N-ad
    ter   % antessive    N-amde
  num
    sing
    plur
  per3
  bool
    plus
    minus
  list
    e_list
    ne_list  hd:bot  tl:list
.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% THEORY %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- tree_extensions.
:- multifile if/2.

% hidden features
hidden_feat(dtrs).

% feature ordering
>>> phon.
num <<< pers.
case <<< num.

%========================================
%       Functional Descriptions
%========================================

% delete(+,+,-)
% delete a bot element from the list and return reduced list

fun del(+,+,-).
del(X,Y,Z) if
   when( (Y=(e_list;ne_list);
          Z=(e_list;ne_list)),
        undelayed_del(X,Y,Z) ).

undelayed_del(El,(list,hd:El,tl:L),L) if true.
undelayed_del(El,(list,hd:H,tl:T1),(list,hd:H,tl:T2)) if del(El,T1,T2).
```

```
%=======================================
%       Import other components
%=======================================
:- ['principles'].

:- ['ps_rules'].

:- ['macros'].

% no lexical rules

:- ['lexicon'].

:- ['test'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% MACROS %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%====================================
%          Synsem Macros
%====================================
%%% noun category, input: case, number
noun_ss(Case-case, Num-num) :=
    (cat,
       head:(noun,
              case:Case,
              num:Num),
       val:e_list).

%%% Verb category, input: Valency list
verb_ss(ConjSt-list) :=
    (cat,
        head:(verb),
        val:ConjSt).

%====================================
%          Word Macros
%====================================
%%% noun word, input: case, number
noun(Case-case, Num-num) :=
    (word,
       cat:(@noun_ss(Case, Num))).

%%% verb word, input: valency list
verb(ConjSt-list) :=
    (word,
       cat:(@verb_ss(ConjSt))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% PHRASE STRUCTURE RULES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%====================================
%       Head Initial PS rule
%====================================
head_init_phrase ##
    (phrase,
     h_init:plus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (Head, h_init:plus),
    cat> (NonHead).
% results head initial phrase
% whose head is also head initial
% i.e. the first word is the head word in the phrase

%====================================
%       Head Final PS rule
%====================================
head_fin_phrase ##
```

```
    (phrase,
     h_init:minus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (NonHead),
    cat> (Head).
% results the head final phrase
% without any constraints on the head

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% PRINCIPLES %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%=====================================
%          Head Feature Principl
%=====================================
phrase
*>
(cat:(head:HF),
 head_dtr:cat:(head:HF)).

%=====================================
%          Valency Principle
%=====================================
phrase
*>
(cat:val:del(NonH,H),
 head_dtr:cat:val:H,
 nonh_dtr:cat:NonH).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% LEXICON %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%====================================
%             Nouns
%====================================
%%% man
kaci    ~~> @noun(nom, sing).
kacma   ~~> @noun(erg, sing).

%%% woman
qali    ~~> @noun(nom, sing).
qalma   ~~> @noun(erg, sing).

%%% book
tcigni      ~~> @noun(nom, sing).
tcigns      ~~> @noun(dat, sing).
tcignebs    ~~> @noun(dat, plur).

%%% letter
tcerili     ~~> @noun(nom, sing).
tcerils     ~~> @noun(dat, sing).

%%% son/daughter
shvili      ~~> @noun(nom, sing).
shvilthan   ~~> @noun(com, sing).

%%% student
studenti    ~~> @noun(nom, sing).
students    ~~> @noun(dat, sing).
studentma   ~~> @noun(erg, sing).
studentebi  ~~> @noun(nom, plur).
studentebs  ~~> @noun(dat, plur).

%%% boy
bitchi  ~~> @noun(nom, sing).
bitchma ~~> @noun(erg, sing).

%%% rope
thoki   ~~> @noun(nom, sing).

%%% fence
```

```
ghobe       ~~> @noun(nom, sing).
ghobidan    ~~> @noun(abl1, sing).
ghobemde    ~~> @noun(ter, sing).

%%% house
saxli       ~~> @noun(nom, sing).
saxlidan    ~~> @noun(abl1, sing).
saxlamde    ~~> @noun(ter, sing).

%%% exam
gamocda     ~~> @noun(nom, sing).
gamocdis    ~~> @noun(gen, sing).
gamocdebis  ~~> @noun(gen, plur).

%==================================
%             Verbs
%==================================

%%% to read
kithxulobs  ~~> @verb([@noun_ss(nom,sing), @noun_ss(dat,_)]).
kithxuloben ~~> @verb([@noun_ss(nom,plur), @noun_ss(dat,_)]).
tcaikithxa  ~~> @verb([@noun_ss(erg,sing), @noun_ss(nom,_)]).

%%% to be afraid of
eshinia     ~~> @verb([@noun_ss(dat,sing), @noun_ss(gen,_)]).
eshiniath   ~~> @verb([@noun_ss(dat,plur), @noun_ss(gen,_)]).

%%% to send
gzavnis     ~~> @verb([@noun_ss(nom,sing), @noun_ss(dat,_), @noun_ss(com,_)]).
gaagzavna   ~~> @verb([@noun_ss(erg,sing), @noun_ss(nom,_), @noun_ss(com,_)]).

%%% to stretch smth. between smth.
gaaba       ~~> @verb([@noun_ss(erg,sing), @noun_ss(nom,_), @noun_ss(abl1,_), @noun_ss(ter,_)]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% TEST SUITE %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%===================================
%      Verb complemeted with nouns
%===================================

t(1, "studenti kithxulobs tcigns",      (phrase, cat:(@verb_ss(e_list))),   1,
      'The student reads the book').
t(2, "studentebi kithxuloben tcignebs", (phrase, cat:(@verb_ss(e_list))),   1,
      'students reads books').
t(3, "studentma tcaikithxa tcigni",     (phrase, cat:(@verb_ss(e_list))),   1,
      'The student read the book').
t(4, "tcigni tcaikithxa studentma",     (phrase, cat:(@verb_ss(e_list))),   1,
      'The student read the book').
t(5, "studentebi kithxuloben",          (phrase, cat:(@verb_ss(ne_list))),  1,
      'Students read').

t(6, "studentebs eshiniath gamocdebis", (phrase, cat:(@verb_ss(e_list))),   1,
      'Students are afraid of exams').
t(7, "students eshinia",                (phrase, cat:(@verb_ss(ne_list))),  1,
      'Student is afraid of').
t(8, "students eshiniath gamocdebis",   bot,                                0,
      'verb noun agreemnet in number').

t(9, "kaci gzavnis tcerils shvilthan",  (phrase, cat:(@verb_ss(e_list))),   1,
      'The man sends the letter tothe son').
t(10, "qalma gaagzavna tcerils",        bot,                            0,
       'wrong case for the noun argumnet').
t(11, "bitchma gaagzavna tcigni",       (phrase, cat:(@verb_ss(ne_list))),  1,
      'the boy sent the book').

t(12, "bitchma gaaba thoki",                    (phrase, cat:(@verb_ss(ne_list))),  1,
      'The boy stretched the rope').
t(13, "kacma thoki gaaba saxlidan ghobemde",    (phrase, cat:(@verb_ss(e_list))),   1,
       'The mam stretched the rope from the house till the fence').
t(14, "saxlidan ghobemde kacma thoki gaaba",    (phrase, cat:(@verb_ss(e_list))),   1,
       'The mam stretched the rope from the house till the fence').
```

```
t(15, "ghobidan saxlamde kacma thoki gaaba",    (phrase, cat:(@verb_ss(e_list))),   1,
      'The mam stretched the rope from the fence till the house').
t(16, "ghobidan gaaba saxlamde kacma thoki",    (phrase, cat:(@verb_ss(e_list))),   1,
      'The mam stretched the rope from the fence till the house').
```

B. GeoGram ver.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% SIGNATURE %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

type_hierarchy
bot
  sign  phon:list  cat:cat  h_init:bool
    word  h_init:plus  arg_st:list
    phrase  head_dtr:sign  nonh_dtr:sign  dtrs:list
  cat  head:head  val:val
  head
    verb  tense:tense
    npn  case:case  num:num  pers:pers
      noun  pers:per3
  val
    frame  expl:list  impl:list
    &list
  case
    nom   % nominative   N-i
    ins   % instrumental N-ith
    gen   % genetive     N-is(si)
    ori   % orientative  N-isken
    abl1  % ablative1    N-idan
    abl2  % ablative2    N-isgan
    ben   % benefactive  N-isthvis
    erg   % ergative     N-ma
    ine   % inessive     N-shi
    dat   % dative       N-s
    loc   % locative     N-ze
    com   % comitative   N-than
    adv   % adverbial    N-ad
    ter   % antessive    N-amde
  tense
    prs
    psc
    pst
    fut
  num
    sing
    plur
  pers
    per1
    per2
    per3
  bool
    plus
    minus
  list
    e_list
    ne_list  hd:bot  tl:list
.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% THEORY %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- tree_extensions.
:- multifile if/2.
:-lex_rule_depth(3).

% hidden features

hidden_feat(dtrs).

% feature ordering
>>> phon.

num <<< pers.
case <<< num.
```

```
%=========================================
%       Functional Descriptions
%=========================================

% append(+,+,-)
% This append assumes that the first or the third argument
% are known to be non_empty or empty lists.

fun append(+,+,-).
append(X,Y,Z) if
   when( (X=(e_list;ne_list);
          Z=(e_list;ne_list))
       , undelayed_append(X,Y,Z)
       ).

undelayed_append(e_list, L, L) if true.
undelayed_append((list, hd:H, tl:T1), L, (list, hd:H, tl:T2)) if append(T1, L, T2).

% delete(+,+,-)
% delete a bot element from the list and return reduced list

fun del(+,+,-).
del(X,Y,Z) if
   when( (Y=(e_list;ne_list);
          Z=(e_list;ne_list)),
        undelayed_del(X,Y,Z) ).

undelayed_del(El,(list,hd:El,tl:L),L) if true.
undelayed_del(El,(list,hd:H,tl:T1),(list,hd:H,tl:T2)) if del(El,T1,T2).

%=========================================
%       Import other components
%=========================================

:- ['principles'].

:- ['ps_rules'].

:- ['macros'].

% no lexical rules

:- ['lexicon'].

:- ['test'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% MACROS %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%=====================================
%          Synsem Macros
%=====================================

%%% noun/pronoun category input: case, number, person
npn_ss(Case-case, Num-num, Per-pers) :=
    (cat,
       head:(npn,
              case:Case,
              num:Num,
              pers:Per),
       val:e_list).

%%% verb category input: explicit_arg_list, Implicit_arg_list, tense
verb_ss(Expl-list, Impl-list, T-tense) :=
    (cat,
        head:(verb,
                tense:T),
        val:(frame,
                expl:Expl,
                impl:Impl)).
```

```
%====================================
%       For Testing and Lexicon
%====================================

%%% noun word, input: case, number
noun(Case-case, Num-num) :=
    (word,
       cat:(cat,
            head:(noun,
                  case:Case,
                  num:Num),
            val:e_list),
       arg_st:e_list).

%%% pronoun (noun) word, input: case, number, person
npn(Case-case, Num-num, Per-pers) :=
    (word,
       cat:(@npn_ss(Case, Num, Per)),
       arg_st:e_list).

%%% verb word, input: explicit_arg_list, implicit_arg_list, tenses
verb(Expl-list, Impl-list, T-tense) :=
    (word,
       cat:(@verb_ss(Expl, Impl, T)),
       arg_st:append(Expl, Impl)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% PHRASE STRUCTURE RULES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%====================================
%       Head Initial PS rule
%====================================

head_init_phrase ##
    (phrase,
     h_init:plus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (Head, h_init:plus),
    cat> (NonHead).
% results head initial phrase
% whose head is also head initial
% i.e. the first word is the head word in the phrase

%====================================
%       Head Final PS rule
%====================================

head_fin_phrase ##
    (phrase,
     h_init:minus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (NonHead),
    cat> (Head).
% results the head final phrase
% without any constraints on the head

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% PRINCIPLES %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%====================================
%          Head Feature Principl
%====================================
phrase
*>
(cat:(head:HF),
 head_dtr:cat:(head:HF)).

%====================================
%          Valency Principle
```

```
%====================================
phrase
*>
((cat:val:(expl:del(NonH, Expl),
           impl:Impl),
  head_dtr:cat:val:(expl:Expl,
                    impl:Impl),
  nonh_dtr:cat:NonH);

 (cat:val:(expl:Expl,
          impl:del(NonH, Impl)),
  head_dtr:cat:val:(expl:Expl,
                    impl:Impl),
  nonh_dtr:cat:NonH)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% LEXICON %%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%====================================
%             Nouns
%====================================

%%% man
kaci    ~~> @noun(nom, sing).
kacma   ~~> @noun(erg, sing).

%%% woman
qali    ~~> @noun(nom, sing).
qalma   ~~> @noun(erg, sing).

%%% book
tcigni      ~~> @noun(nom, sing).
tcigns      ~~> @noun(dat, sing).
tcignebs    ~~> @noun(dat, plur).

%%% letter
tcerili     ~~> @noun(nom, sing).
tcerilebi   ~~> @noun(nom, plur).
tcerils     ~~> @noun(dat, sing).

%%% son/daughter
shvili      ~~> @noun(nom, sing).
shvilthan   ~~> @noun(com, sing).
shvilebthan ~~> @noun(com, plur).

%%% student
studenti    ~~> @noun(nom, sing).
students    ~~> @noun(dat, sing).
studentma   ~~> @noun(erg, sing).
studentebi  ~~> @noun(nom, plur).
studentebs  ~~> @noun(dat, plur).
studentebthan   ~~> @noun(com, plur).

%%% boy
bitchi  ~~> @noun(nom, sing).
bitchma ~~> @noun(erg, sing).

%%% rope
thoki   ~~> @noun(nom, sing).
thoks   ~~> @noun(dat, sing).

%%% fence
ghobe       ~~> @noun(nom, sing).
ghobidan    ~~> @noun(abl1, sing).
ghobemde    ~~> @noun(ter, sing).

%%% house
saxli       ~~> @noun(nom, sing).
saxlidan    ~~> @noun(abl1, sing).
saxlamde    ~~> @noun(ter, sing).

%%% exam
```

```
gamocda     ~~> @noun(nom, sing).
gamocdis    ~~> @noun(gen, sing).
gamocdebis  ~~> @noun(gen, plur).


%===================================
%                VERBS
%===================================

%%% to read smth
kithxulobs  ~~> @verb([@npn_ss(nom,sing,per3), @npn_ss(dat,_,per3)],  [], prs).
kithxuloben ~~> @verb([@npn_ss(nom,plur,per3), @npn_ss(dat,_,per3)],  [], prs).
tcaikithxa  ~~> @verb([@npn_ss(erg,sing,per3), @npn_ss(nom,_,per3)],  [], pst).
vkithxulobdith  ~~> @verb([@npn_ss(nom,plur,per1), @npn_ss(dat,_,per3)],  [], psc).

%%% to be afraid of smth./smb.
eshinia     ~~> @verb([@npn_ss(dat,sing,per3)],  [@npn_ss(gen,_,_)], prs).
eshiniath   ~~> @verb([@npn_ss(dat,plur,per3)],  [@npn_ss(gen,_,_)], prs).
meshinia    ~~> @verb([@npn_ss(dat,sing,per1)],  [@npn_ss(gen,_,_)], prs).
shegeshinda ~~> @verb([@npn_ss(dat,sing,per2)],  [@npn_ss(gen,_,_)], pst).

%%% to send smth./smb. smwh.
gzavnis     ~~> @verb( [@npn_ss(nom,sing,per3), @npn_ss(dat,_,per3)],
                       [@npn_ss(com,_,_)], prs).
gaagzavna   ~~> @verb( [@npn_ss(erg,sing,per3), @npn_ss(nom,_,per3)],
                       [@npn_ss(com,_,_)], pst).
gavagzavne  ~~> @verb( [@npn_ss(erg,sing,per1), @npn_ss(nom,_,per3)],
                       [@npn_ss(com,_,_)], pst).
agzavnith   ~~> @verb( [@npn_ss(nom,plur,per2), @npn_ss(dat,_,per3)],
                       [@npn_ss(com,_,_)], prs).

%%% to send for smb. smth. smwh.
gagigzavne  ~~> @verb( [@npn_ss(erg,sing,per1), @npn_ss(dat,sing,per2), @npn_ss(nom,_,per3)],
                       [@npn_ss(com,_,_)], pst).

%%% to stretch smth. between smth.
gaaba       ~~> @verb( [@npn_ss(erg,sing,per3), @npn_ss(nom,_,per3)],
                       [@npn_ss(abl1,_,_), @npn_ss(ter,_,_)], pst).

%%% to stretch for smb. smth. between smth.
gagibav     ~~> @verb( [@npn_ss(nom,sing,per1), @npn_ss(dat,sing,per2), @npn_ss(dat,_,per3)],
                       [@npn_ss(abl1,_,_), @npn_ss(ter,_,_)], fut).

%===================================
%                PRONOUNS
%===================================

%%% 1st person singular
me ~~>          @npn((nom;erg;dat), sing, per1).
chemi ~~>       @npn(gen,          sing, per1).
chemidan ~~>    @npn(abl1,         sing, per1).
chemgan ~~>     @npn(abl2,         sing, per1).
chemthan ~~>    @npn(com,          sing, per1).
chemamde ~~>    @npn(ter,          sing, per1).
%%% 1st person plural
chven ~~>       @npn((nom;erg;dat), plur, per1).
chveni ~~>      @npn(gen,          plur, per1).
chvenidan ~~>   @npn(abl1,         plur, per1).
chvengan ~~>    @npn(abl2,         plur, per1).
chventhan ~~>   @npn(com,          plur, per1).
chvenamde ~~>   @npn(ter,          plur, per1).

%%% 2nd person singular
shen ~~>        @npn((nom;erg;dat), sing, per2).
sheni ~~>       @npn(gen,          sing, per2).
shenidan ~~>    @npn(abl1,         sing, per2).
shengan ~~>     @npn(abl2,         sing, per2).
shenthan ~~>    @npn(com,          sing, per2).
shenamde ~~>    @npn(ter,          sing, per2).
%%% 2nd person plural
thqven ~~>      @npn((nom;erg;dat), plur, per2).
thqveni ~~>     @npn(gen,          plur, per2).
thqvenidan ~~>  @npn(abl1,         plur, per2).
```

```
thqvengan ~~>   @npn(abl2,         plur, per2).
thqventhan ~~>  @npn(com,          plur, per2).
thqvenamde ~~>  @npn(ter,          plur, per2).

%%% 3rd person singular
is ~~>          @npn(nom,          sing, per3).
igi ~~>         @npn(nom,          sing, per3).
mas ~~>         @npn(dat,          sing, per3).
man ~~>         @npn(erg,          sing, per3).
misith ~~>      @npn(ins,          sing, per3).
misi ~~>        @npn(gen,          sing, per3).
misgan ~~>      @npn((abl2;abl1),  sing, per3).
masthan ~~>     @npn(com,          sing, per3).
%%% 3rd person plural
isini ~~>       @npn(nom,          plur, per3).
math ~~>        @npn((erg;dat),        plur, per3).
mathi ~~>       @npn(gen,          plur, per3).
mathgan ~~>     @npn((abl1;abl2),  plur, per3).
maththan ~~>    @npn(com,          plur, per3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% TEST SUITE %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%=====================================
%      Verb complemeted with nouns
%=====================================

t(1, "studenti kithxulobs tcigns",      (phrase, cat:(@verb_ss(e_list, e_list, prs))),  1,
     'The student reads the book').
t(2, "studentebi kithxuloben tcignebs", (phrase, cat:(@verb_ss(e_list, e_list, prs))),  1,
     'students reads books').
t(3, "studentma tcaikithxa tcigni",     (phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
     'The student read the book').
t(4, "tcigni tcaikithxa studentma",     (phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
     'The student read the book').
t(5, "studentebi kithxuloben",          (phrase, cat:(@verb_ss(ne_list, e_list, prs))), 1,
     'Students read').
t(6, "studentebs eshiniath gamocdebis", (phrase, cat:(@verb_ss(e_list, e_list, prs))),  1,
     'Students are afraid of exams').
t(7, "students eshinia",                (phrase, cat:(@verb_ss(e_list, ne_list, prs))), 1,
     'Student is afraid of').
t(8, "students eshiniath gamocdebis",   bot,                                            0,
     'verb noun agreemnet in number').

t(9, "kaci gzavnis tcerils shvilthan",  (phrase, cat:(@verb_ss(e_list, e_list, prs))),  1,
     'The man sends the letter tothe son').
t(10, "qalma gaagzavna tcerils",       bot,                                            0,
      'wrong case for the noun argumnet').
t(11, "bitchma gaagzavna tcigni",      (phrase, cat:(@verb_ss(e_list, ne_list, pst))), 1,
      'the boy sent the book').

t(12, "bitchma gaaba thoki",                (phrase, cat:(@verb_ss(e_list, ne_list, pst))), 1,
      'The boy stretched the rope').
t(13, "kacma thoki gaaba saxlidan ghobemde",(phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
      'The mam stretched the rope from the house till the fence').
t(14, "saxlidan ghobemde kacma thoki gaaba",(phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
      'The mam stretched the rope from the house till the fence').
t(15, "ghobidan saxlamde kacma thoki gaaba",(phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
      'The mam stretched the rope from the fence till the house').
t(16, "ghobidan gaaba saxlamde kacma thoki",(phrase, cat:(@verb_ss(e_list, e_list, pst))),  1,
      'The mam stretched the rope from the fence till the house').

%=====================================
%      Verb complemeted with nouns and pronouns
%=====================================

t(17, "chven tcignebs vkithxulobdith",(phrase, cat:(@verb_ss(e_list, e_list, psc))),    1,
      'We were reading books').
t(18, "meshinia sheni",              (phrase, cat:(@verb_ss(ne_list, e_list, prs))),   1,
      'I am afraid of you').
t(19, "me meshinia sheni",           (phrase, cat:(@verb_ss(e_list, e_list, prs))),    1,
      'I am afraid of you').
```

```
t(20, "meshinia misi",                (phrase, cat:(@verb_ss(ne_list, e_list, prs))),   1,
      'I am afraid of him').
t(21, "shen shegeshinda",             (phrase, cat:(@verb_ss(e_list, ne_list, pst))),   1,
      'you got frightened').
t(22, "me gavagzavne shen",           bot,                                              0,
      '3rd person required 2nd person got').
t(23, "gavagzavne tcerili shvilthan", (phrase, cat:(@verb_ss(ne_list, e_list, pst))),   1,
      'I sent a letter to to the son').
t(24, "thqven agzavnith tcerils",     (phrase, cat:(@verb_ss(e_list, ne_list, prs))),   1,
      'You-pl are sending a letter').
t(25, "gagibav saxlamde thoks",       (phrase, cat:(@verb_ss(ne_list, ne_list, fut))),  1,
      'I will stretch for you the rope till the house').
t(26, "gagigzavne tcerilebi studentebthan",     (phrase, cat:(@verb_ss(ne_list, e_list, pst))), 1,
      'I sent for you letters to students').
```

C. GeoGram ver.3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% SIGNATURE %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

type_hierarchy
bot
  sign  phon:list  cat:cat  h_init:bool
    word  h_init:plus  arg_st:list
    phrase  head_dtr:sign  nonh_dtr:sign  dtrs:list
      ch_phrase %tech
      ah_phrase %tech
      hc_phrase %tech
  cat  head:head  val:val
  head
    verb  tense:tense
    nominal  case:case
      npn  num:num  pers:pers
        &poss_npn
        arg_npn  case:arg_c
          &arg_noun
        noun  pers:per3
          arg_noun
          &poss_noun
      adjunct  case:adj_c  mod:ne_list
        poss_npn
          poss_pn  case:non_cnst
          poss_noun  case:cnst %tech, cnst
        qnt
          qnt_i  case:non_cnst
          qnt_  case:cnst
        adj
          adj_i  case:non_cnst
          adj_  case:cnst
  val
    frame  expl:list  impl:list
    comps  spec:spec  comp:list
    &e_list
  spec
    unspec
    specif
      attrib
      deter
        n_poss
        pnp_q
          quant
          pn_poss
    non_deter %tech
          &unspec
          &attrib
  case
    arg_c
      case_i
        nom   % nominative   N-i
        ins   % instrumental N-ith
        gen   % genetive     N-is(si)
        ori   % orientative  N-isken
        abl1  % ablative1    N-idan
        abl2  % ablative2    N-isgan
        ben   % benefactive  N-isthvis
      erg   % ergative     N-ma
      case_
        ine   % inessive     N-shi
        dat   % dative       N-s
        loc   % locative     N-ze
        com   % comitative   N-than
        adv   % adverbial    N-ad
        ter   % antessive    N-amde
    adj_c
      non_cnst
        nom_c
        dat_c
        erg_c
      cnst
    cnst_i  %tech
      &cnst
      &case_i
  tense
    prs
    psc
    pst
    fut
  num
    sing
    plur
  pers
    per1
    per2
    per3
  bool
    plus
    minus
  list
    e_list
    ne_list  hd:bot  tl:list
.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% THEORY %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- tree_extensions.
:- multifile if/2.
:-lex_rule_depth(3).

% hidden features

hidden_feat(dtrs).

% feature ordering
>>> phon.
num <<< pers.
case <<< num.

%=========================================
%       Functional Descriptions
%=========================================

% append(+,+,-)
% input: two lists
% output: concatenation of the lists
% This append assumes that the first or the third argument
% are known to be non_empty or empty lists.

fun append(+,+,-).
append(X,Y,Z) if
   when( (X=(e_list;ne_list);
          Z=(e_list;ne_list))
       , undelayed_append(X,Y,Z)
       ).

undelayed_append(e_list, L, L) if true.
undelayed_append((list, hd:H, tl:T1), L, (list, hd:H, tl:T2)) if append(T1, L, T2).

% delete(+,+,-)
% delete a bot element from the list and return reduced list
% input: element and list
% output: list minus element

fun del(+,+,-).
del(X,Y,Z) if
   when( (Y=(e_list;ne_list);
          Z=(e_list;ne_list)),
        undelayed_del(X,Y,Z) ).
undelayed_del(El,(list,hd:El,tl:L),L) if true.
undelayed_del(El,(list,hd:H,tl:T1),(list,hd:H,tl:T2)) if del(El,T1,T2).

% mod_case(+,+,-)
% input: adjunct case
% output: modifiers case

fun mod_case(+,-).
mod_case(erg_c, erg) if true.
mod_case(dat_c, case_) if true.
mod_case(cnst, case) if true.
mod_case(nom_c, cnst_i) if true.

% specify(+,+,-)
% input: the specification function of the adjunct and
%        the specification value og the head
% output: specification value of the phrase

fun specify(+,+,-).
specify((S,specif), unspec, S) if true.
specify((S,specif), attrib, S) if true.
specify(n_poss, quant, n_poss) if true.
specify(pn_poss, quant, pnp_q) if true.
specify(quant, pn_poss, pnp_q) if true.


%========================================
%       Import other components
%========================================

:- ['principles'].

:- ['ps_rules'].

:- ['macros'].

% no lexical rules

:- ['lexicon'].

:- ['test'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% Macros %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%=====================================
%          Synsem Macros
%=====================================

%%% noun category input: case, number, specification, complement
%%% used to complect the argument structure of adjuncts
noun_ss(Case-case, Num-num, Spec-spec, Comp-list) :=
    (cat,
       head:(noun,
              case:Case,
              num:Num),
       val:(spec:Spec,
            comp:Comp)).

%%% noun complement input: number
%%% used for noun complements
n_comp_ss(Num-num) :=
    (cat,
       head:(noun,
              case:cnst,
              num:Num),
       val:spec:non_deter).

%%% argument noun/pronoun category input: case, number, person
%%% used to complect the argument structure of the verb
arg_npn_ss(Case-arg_c, Num-num, Per-pers) :=
    (cat,
       head:(arg_npn,
              case:Case,
              num:Num,
              pers:Per),
       val:comps).

%%% verb category input: explicit_arg_list, Implicit_arg_list, tense
%%% used in test
verb_ss(Expl-list, Impl-list, T-tense) :=
    (cat,
        head:(verb,
                tense:T),
        val:(frame,
                expl:Expl,
                impl:Impl)).


%=====================================
%       For Testing and Lexicon
%=====================================

%%% noun word, input: case, number, complement
%%% used in lexicon for arg-nouns and poss-nouns
noun(Case-case, Num-num, Comp-list) :=
    (word,
       cat:(@noun_ss(Case, Num, unspec, Comp)),
       arg_st:Comp).

%%% pronoun word, input: case, number, person
%%% used in lexicon for possesive and argument pronouns
pn(Case-case, Num-num, Per-pers) :=
    (word,
       cat:(head:(npn,
                    case:Case,
                    num:Num,
                    pers:Per),
            val:(spec:n_poss,
                comp:e_list)),
       arg_st:e_list).

%%% verb word, input: explicit_arg_list, implicit_arg_list, tenses
%%% used in lexicon
verb(Expl-list, Impl-list, T-tense) :=
    (word,
       cat:(@verb_ss(Expl, Impl, T)),
       arg_st:append(Expl, Impl)).

%%% adjective word, input: case
%%% used in lexicon
adj(Case-adj_c) :=
    (word,
       cat:(cat,
            head:(adj,
                    case:Case,
                    mod:[@noun_ss(mod_case(Case),_,_,_)]),
                    % constraints are relaxed on spec feature
                    % to avoid inequations
            val:e_list),
       arg_st:e_list).

%%% quantifier word, input: case
%%% used in lexicon
qnt(Case-adj_c) :=
    (word,
       cat:(cat,
            head:(qnt,
                    case:Case,
                    mod:[@noun_ss(mod_case(Case),sing,_,_)]),
                    % constraints are relaxed on spec feature
                    % to avoid inequations
            val:e_list),
       arg_st:e_list).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% PHRASE STRUCTURE RULES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%=====================================
%       Head Initial Verb PS Rule
%=====================================

head_init_phrase ##
    (hc_phrase,
     h_init:plus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (Head, h_init:plus,
                cat:head:verb),
    cat> (NonHead).
% results head initial verb phrase
% whose head is also head initial
% i.e. the first word is the head word in the phrase

%=====================================
%       Head Final Verb PS Rule
%=====================================

head_fin_phrase ##
    (ch_phrase,
     h_init:minus,
     head_dtr:Head,
     nonh_dtr:NonHead)
===>
    cat> (NonHead),
    cat> (Head, cat:head:verb).
% results the head final verb phrase
% without any constraints on the head

%=====================================
%       Adjunct-Noun PS Rule
%=====================================

adjunct_noun ##
    (ah_phrase,
       h_init:minus,
       cat:val:(spec:Spec,
                comp:e_list),
       head_dtr:Head,
       nonh_dtr:NonHead)
===>
    cat> (NonHead, cat:head:(adjunct, POS,
                                  mod:[H_cat])),

    cat> (Head, cat:(H_cat, head:noun,
                     val:spec:Hspec)),

    goal> ( (POS = adj, Spec = specify(attrib,Hspec));
            (POS = qnt, Spec = specify(quant,Hspec));
            (POS = poss_pn, Spec = specify(pn_poss,Hspec));
            (POS = poss_noun,  Spec = specify(n_poss, Hspec))
          ).
% Depending on adjunct type the phrase changes the Spec value
% Resulted phrase can not take any complements

%=====================================
%       Complement-Noun PS Rule
%=====================================

complement_noun ##
    (ch_phrase,
       h_init:minus,
       cat:val:(spec:unspec,
                comp:e_list),
       head_dtr:Head,
       nonh_dtr:NonHead)
```

```
===>
    cat> (NonHead, cat:Comp_ss),

    cat> (Head, cat:(head:noun,
                     val:comp:[Comp_ss])).
% The cat of complement is encoded in complement list of the head
% Resulted phrase is unspecified and has empty list of complements

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% PRINCIPLES %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%=====================================
%       Head Feature Principl
%=====================================

phrase
*>
(cat:(head:HF),
 head_dtr:cat:(head:HF)).


%=====================================
%       Verb Valency Principle
%=====================================

(phrase, cat:head:verb)
*>
((cat:val:(expl:del(NonH, Expl),
           impl:Impl),
  head_dtr:cat:val:(expl:Expl,
                    impl:Impl),
  nonh_dtr:cat:NonH);

 (cat:val:(expl:Expl,
          impl:del(NonH, Impl)),
  head_dtr:cat:val:(expl:Expl,
                    impl:Impl),
  nonh_dtr:cat:NonH)).

%=====================================
%       We skip the lexicon and the test suite due to its length
%       They can be found along with other implemented grammars at:
%       https://sites.google.com/site/lashabzianidze/thesis
%=====================================
```

D. Lexical rules for nominals

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Lexical Rules %%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%####################################################################
%        Lexical Rules For Nominals (Noun, Adjective)
%####################################################################

%=======================================
%    Noun Lexeme Pluralization
%=======================================
% noun lexeme ---> noun plural lexeme
%---------------------------------------
pluralization  ##

    (L, (@lex_n(Par, plus, Compl)))
**>
    (@lex_n(decl_i, both, Compl), init:L)

if  (Par=(decl_i;decl_a) -> (a_ Para)=(a_ ai); (a_ Para)=(a_ eou))

morphs

(_,'_',NonFullSt)   becomes (NonFullSt,eb,'_',NonFullSt,eb) when (Para = ai),
(FullSt,'_',_)      becomes (FullSt,eb,'_',FullSt,eb)       when (Para = eou).

%=======================================
%      Noun Declension in Logical Cases
%=======================================
% noun lexeme ---> 14 word forms marked by logical cases
%---------------------------------------
logic_decl  ##

    (L, @lex_n(Par, Pl, Compl))
**>
    (@noun(Case, Num, Compl), lex:L)

if  isLogCase(Case, a_ C),
    isParadigm(Par, a_ P),
    (Pl=(plus;minus) -> Num=sing; Num=plur)

morphs

(FullSt,'_',_)      becomes (FullSt,i)          when (C = nom, P = decl_i),
(FullSt,'_',_)      becomes (FullSt)            when (C = nom),

(FullSt,'_',_)      becomes (FullSt,ma)         when (C = erg, P = decl_i),
(FullSt,'_',_)      becomes (FullSt,m)          when (C = erg),

(FullSt,'_',_)      becomes (FullSt,s)          when (C = dat),

(FullSt,'_',_)      becomes (FullSt,shi)        when (C = ine),

(FullSt,'_',_)      becomes (FullSt,ze)         when (C = loc),

(FullSt,'_',_)      becomes (FullSt,than)       when (C = com, P = decl_i),
(FullSt,'_',_)      becomes (FullSt,sthan)      when (C = com),

(_,'_',NonFullSt)   becomes (NonFullSt,ad)      when (C = adv, (P = decl_i; P = decl_a)),
(FullSt,'_',_)      becomes (FullSt,d)          when (C = adv),

(_,'_',NonFullSt)   becomes (NonFullSt,amde)    when (C = ter, (P = decl_i; P = decl_a)),
(FullSt,'_',_)      becomes (FullSt,mde)        when (C = ter),

(FullSt,'_',_)      becomes (FullSt,thi)        when (C = ins, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,ith)     when (C = ins),

(FullSt,'_',_)      becomes (FullSt,si)         when (C = gen, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,is)      when (C = gen),
(FullSt,'_',_)      becomes (FullSt,sken)       when (C = ori, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,isken)   when (C = ori),

(FullSt,'_',_)      becomes (FullSt,dan)        when (C = abl1, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,idan)    when (C = abl1),

(FullSt,'_',_)      becomes (FullSt,sgan)       when (C = abl2, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,isgan)   when (C = abl2),

(FullSt,'_',_)      becomes (FullSt,sthvis)     when (C = ben, P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,isthvis) when (C = ben).

%=======================================
%    Marking Noun by Possessive Case
%=======================================
% noun word form ---> the same noun marked with possessive case
%---------------------------------------
poss_decl  ##

    (L, @lex_n(Par, Pl, Compl))
**>
    (@noun(cnst, Num, Compl), cat:head:mod:[head:noun], lex:L)

if  isParadigm(Par, a_ P),
    (Pl=(plus;minus) -> Num=sing; Num=plur)

morphs

(FullSt,'_',_)      becomes (FullSt,s)          when (P = decl_ou),
(_,'_',NonFullSt)   becomes (NonFullSt,is).

%=======================================
%  Adjective/Quantifier Declension in Logical Cases
%=======================================
%  adjective/quantifier lexeme ---> 3/1 word forms marked by logical cases
%---------------------------------------
adjunct_decl  ##

    (L, (adj_lex;qnt_lex))
**>
    (Adjunct, lex:L)

if (L=adj_lex -> Adjunct=(@adj(Case)); Adjunct=(@qnt(Case))),
   ((Case=erg_c), (a_ C)=(a_ erg_c);
    (Case=dat_c), (a_ C)=(a_ dat_c);
    (Case=nom_c), (a_ C)=(a_ nom_c);
    (Case=cnst),  (a_ C)=(a_ cnst))

morphs

(Stem,i,'_',_)      becomes (Stem,ma)   when (C=erg_c),
(Stem,i,'_',_)      becomes (Stem)      when (C=dat_c),
(Stem,i,'_',_)      becomes (Stem,i)    when (C=nom_c),
(Stem,[A],'_',_)    becomes (Stem,A)    when (C=cnst, A\=i).

%=======================================
%  Normalization of Adjective/Qiantifier Lexeme
%=======================================
% adj_lexeme/qnt_lexeme ---> noun_lexeme
%---------------------------------------
normalizarion  ##

    (L, (@lex_a(Decl);@lex_q(Decl)) )
**>
    (@lex_n(Decl,Plural,e_list), init:L)

if  (L=adj_lex -> Plural=plus; Plural=minus)

morphs

(Stem,i,'_',PlStem)     becomes (Stem,'_',PlStem),
(Stem,'_',PlStem)       becomes (Stem,'_',PlStem).
```

```
%=====================================
%       We skip the other parts of the grammar for its length
%       All further inforation about the grammar can be found at:
%       https://sites.google.com/site/lashabzianidze/thesis
%=====================================
```