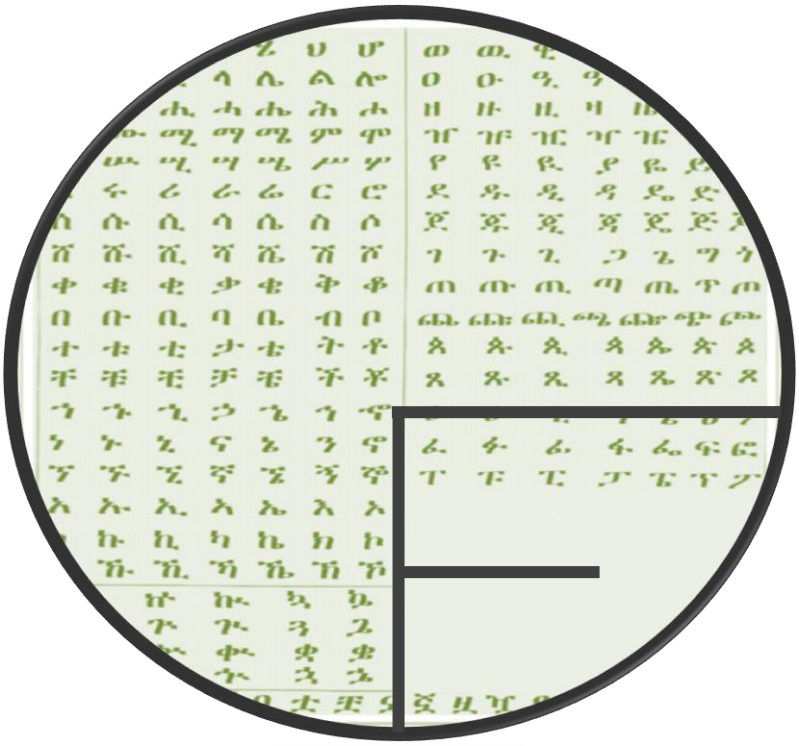
CHALMERS



IMPLEMENTING AN OPEN SOURCE AMHARIC RESOURCE GRAMMAR IN GF

Master of Science Thesis in Intelligent Systems Design

MARKOS KASSA GOBENA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2010

IMPLEMENTING AN OPEN SOURCE AMHARIC RESOURCE GRAMMAR IN GF

MARKOS KASSA GOBENA

Examiner: AARNE RANTA (Prof.)
Supervisor: RAMONA ENACHE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Amharic Fidäl and the GF official logo marking the *implementation of Amharic resource grammar in GF*

Department of Computer Science and Engineering
Göteborg, Sweden November 2010

Abstract

Developing language applications or localization of software is a resource intensive task that requires the active participation of stakeholders with various backgrounds. With a constant increase in the amounts of electronic information and the diversity of languages which are used to produce them, these challenges get compounded. Various researches in the fields of computational linguistics and computer science have been carried out while still many more are on their way to alleviate such problems. Grammatical Framework (GF) is one potential candidate to this. GF is a grammar formalism designed for multilingual grammars. A multilingual grammar has a shared representation, called abstract syntax, and a set of concrete syntaxes that map the abstract syntax to different languages. In this thesis, we describe an implementation of Amharic, a Semitic language spoken in Ethiopia, as a resource grammar in GF and we deal with orthography, morphology and syntax of the language. The work contributes to the reduction of the amount of time and energy spent while developing language-related applications using Amharic.

This report is written in English.

Acknowledgement

I am humbly grateful to my Lord for guiding me and helping me all the way through. My heartfelt thanks goes to my examiner and father of GF Aarne Ranta, for letting me undertake this project and showing me lots of kindness whenever I visited his office. Ramona Enache, you have always been brilliant and attentive. I sincerely thank you very much for the supervision and creating a friendly environment whenever we sit for discussions. I love my mom so I can't thank her enough for everything. Thank you brothers and sisters for all the support and love I got while trying to make it on a foreign soil. There were times when I was flying in the air without my compasses set, thank you Selam for showing me the directions, MLL. Dear friends; thanks for making my life so rich. Gunilla, you have all been a God-sent. Gossish & family, it always felt like home-away-from-home whenever I visited your place. The Ethiopian community at St. Gabriel Ethiopian Orthodox Church, my indebtedness goes to you for the love and support you gave me, may God bless our gathering forever and ever amen!

Contents

List of Abbreviations

API	Application Programmers Interface
DEF	Defininte
EU	European Union
FEM	Feminine
GF	Grammatical Framework
GNU	GNU is Not Unix
GPL	General Public License
LGPL	Lesser General Public License
MASC	Masculine
MOLTO	Multilingual Online Translation
NLP	Natural Language Processing
NP	Noun Phrase
PGF	Portable Grammar Format
RGL	Resource Grammar Library
SERA	System for Ethiopic Representation in ASCII
SOV	Subject-Object-Verb
TAM	Tense-Aspect-Mood
UTF-8	8-bit Unicode Transformation Format
VP	Verb Phrase

Chapter 1

Introduction

One of the fundamental features of human behavior is the natural language. It is a vital component through which we communicate about the world that affects our daily lives. Most human knowledge is recorded using natural languages, therefore, only computers that have the capability to understand natural language can access the information contained in the natural language efficiently.

Natural language processing (NLP) can be described as the ability of computers to generate and interpret natural languages. NLP is also a major subfield of study in computer science. The applications that will be possible when NLP capabilities are fully realized are impressive as computers would be able to understand and process natural language, translate languages accurately and in real time, or extract and summarize information from a variety of data sources, depending on the users' requests. (Grishman, 1994)

Language engineering is another topic that has attracted the attention of both linguists and computer scientists who are involved in NLP. This venture effort aspires to bring a computation based representation of human language enabling further processing. One feasible approach to this is the formalization of linguistic knowledge into some form of grammatical rules. The appropriate term to describe such an approach is called *grammar formalism.*

Grammatical Framework (GF)[1] is one such grammar formalism which is based on constructive type theory to express the semantics of natural languages for multilingual grammar applications. (Ranta, 2004)

This framework (GF) has a language library known as the '*GF Resource Grammar Library'* which is constituted of resource grammars implemented using the GF programming language (Ranta, 2009) for various languages. The resource grammar for each language defines a complete set of morphological paradigms and a syntax fragment. It is composed of a common

[1] http://www.grammaticalframework.org

representation, called *abstract syntax*, and a set of *concrete syntaxes*. These two are distinctly identified in GF. The first represents the structure or meaning of values in the language while the later describes their appearance. The idea is that the abstract syntax is kept off from irrelevant details and concentrates on the structure of the common features. The concrete syntaxes, on the other hand, handle the details of language specific decisions and are written as linearization rules for the abstract terms. Therefore, in GF context, compilation is nothing but parsing with the concrete syntax of source language and linearizing the resulting tree into the target language. The advantage of such a design is that the abstract syntax can have several concrete linearizations allowing it to work like an interlingua between the concrete syntaxes.

These modules also have the capabilities of sharing code through inheritance while still abstracting information. The inclusion of such important software engineering concepts in GF grammars proves an increasing work efficiency as labor would be divided between grammarians working on different modules. This effect is specially magnified while implementing natural languages of similar family, such as a Romance or Scandinavian that could share nearly three-fourth of their implementation code.

The application grammarians make use of the resource grammars through application programming interfaces (API's) included as abstract syntaxes in the GF library. Therefore, the overall aim of such a library based design is to make it possible for linguistically untrained application programmers to write linguistically correct application grammars encoding the semantics of special domain as stated by Ranta in (Ranta, 2009)

The ongoing GF resource grammar project provides resource grammars for seventeen languages, Amharic being the eighteenth. More languages are also under construction when we write this report. Currently there are also a good number of applications that use GF which include the verification tool KeY[1], the dialogue system research project TALK[2] and the educational project WebALT[3] are major ones. The GF inspired EU project, MOLTO[4] is another ambitious initiative that develops a set of tools for translating texts between multiple languages in real time with high quality. Moreover, the availability of such a library as open-source software under the GNU LGPL helps the aspirations being made to bring less recognized and NLP wise under resourced languages, such as Amharic, to the world of computation.

Motivation

Amharic (አማርኛ amarəññña) is a Semitic language spoken in North and Central Ethiopia. It is the second most-spoken Semitic language in the world, after Arabic, and the official working

[1] http://www.key-project.org
[2] http://www.talk-project.org
[3] http://webalt.math.helsinki.fi
[4] http://www.molto-project.eu

language of the Federal Democratic Republic of Ethiopia[1]. Currently, there is almost no software or web-services that are used for implementing language specific features such as spell checking, grammar, translation etc for Amharic. Amharic is not even part of the free online translator from Google™, Google-Translate™. The availability of Google™ search and G-mail™ electronic mail software in Amharic are the two recent progresses of the language in the cyber world. It can therefore be concluded that there is only little work done on computational resources for Amharic and its use for parsing and generation. In addition we strongly share the conclusion by Gasser (Gasser, 2010) that within the current explosion in the quantity of information and in the means to access it, much of the world has been left behind because the information is not in a language that they understand. This fact, of all the rest, motivated us to contribute to the research in NLP of Amharic by implementing a computational grammar as a resource in the GF library. By doing so, we believe, our work gives yet another perspective for current research and strengthens the attempts already made on the NLP of Amharic.

Organization of the Report

The next chapter discusses the grammatical framework and some important aspects of the formalism. The third chapter describes the Amharic grammar. Included in the discussion are highlights of the different word classes, phrase structures and sentence structures of the language which we provide through an example. The chapter that follows details the main product of this thesis: the implementation of Amharic in the GF library and the various decisions taken while designing the modules. The final chapters deal with related and future works plus conclusions and recommendations we have made with regard to our work.

[1] http://www.en.wikipedia.org/wiki/Amharic

Chapter 2

Grammatical Framework

This chapter gives a thumbnail introduction to GF and discusses some of the points that are crucial in resource grammar engineering.

2.1 Multilingual Grammars

Simply defined, a 'multilingual grammar' is a grammar that describes multiple languages that share a common representation. (Ranta, 2009) Various translation and localization applications between languages make use of multilingual grammars. The common feature, which is shared by the languages under question, alleviates the load of work needed to implement another new language.

To better show the concepts lying behind multilingual grammars with a shared representation, we give the following explanation by Ranta in his description of multilingual grammar engineering (Ranta, 2009).

'Formally, a multilingual grammar in GF is a pair

G = < A, {C1, . . . ,Cn} >

where **A** *is an abstract syntax and* **Ci** *are concrete syntaxes for* **A**. *The abstract syntax is a collection of categories and functions. It defines a tree language, which is a common representation of the string languages defined by the concrete* syntaxes. *A concrete syntax is given by a method of linearization, which translates abstract syntax trees into strings (more generally, into records of strings and features). A multilingual grammar thereby defines a system of multilingual generation from the shared abstract syntax.'*

Ranta goes on to explain the important property of GF grammars that emanates from the fact of the generations being the primary directions of grammatical description, a different feature from other formalisms. Furthermore the parser can independently map strings directly into abstract syntax trees making the linearization inevitable.

2.2 Translation and GF

From the forgoing explanation it is logical to deduce that, in the context of multilingual grammars, translation is nothing but the parsing from one language followed by linearization into another. Transfer functions need not be required between the two languages under translation question; instead, the abstract syntax takes care of the interlingua burden doing the magic work.

It would of course be unrealistic of us if we claimed that GF already has a unique single interlingua to get the translation work done. Rather, the matter worth noting is that GF is just a framework for giving interlinguas i.e. language-independent *abstract syntaxes* that deal with pure tree structures and the set of language-dependent *concrete syntaxes* that specify how the interlinguas are rendered in different languages.

In addition, the interlingua assumes various semantic descriptions according to the application domain it is built for, this eventually provides a meaning-wise-intact translation in that specific domain.

2.3 Application Grammars and Resource Grammars

The narrower the scope of the domain that we consider for translation, the better the semantics of the translation becomes. But this can be a resource intensive operation requiring the attendance of domain experts and linguists alike. The knowledge of the domain experts that is required for the translation work varies according to the domain under consideration while the same linguistic expertise is needed for each domain. However, it is often very rare and resource demanding per se, to bring the two kind of expertise around the same table. This is when GF comes to the rescue by providing a division of labor in grammar engineering between domain experts and linguistic experts. Such a fine division is brought about by the distinctions made between application grammars and resource grammars.

Ranta in the same document goes on to state that '*an application grammar has an abstract syntax expressing the semantics of an application domain. A resource grammar has an abstract syntax expressing linguistic structures. The concrete syntax of an application grammar can be defined as a mapping to the abstract syntax of the resource grammar: it tells what structures are used for expressing semantic object, instead of telling what strings are used*.'

2.4 The Resource Grammar Library

As we tried to highlight in the first chapter of the report, resource grammars in GF technically serve as standard software engineering libraries like those found in main stream programming languages such as Java and C. The abstract syntax of the resource grammar is the API of the

library through which application grammarians can access it. The rest of the concrete syntax is abstracted away from the application grammarian and is the duty of the resource grammarian.

Orchestrating the design into standard libraries can save the man-power required for coding and betters the quality of work by allocating different subtasks of grammar writing to different grammarians. Such shared representations of the resource library are crucial for the application grammarians too as it makes their life easier while they port applications from one language to another.

Chapter 3

Background

In this chapter we summarize the main features of the Amharic language. The majority of the concepts are adopted from a grammar book written in Amharic (Yimam, 1987). Our intention is neither to give the details of the grammar nor provide a crash course of the language. The works by Atalech (Argaw, 2002) and Abiyot (Bayou, 2000) discuss such concise introduction to the grammar of Amharic.

3.1 Amharic

Amharic /አማርኛ is a member of the Semitic branch of the Afro-Asiatic language family. It is spoken by over twenty five millions people and is the working language of the government of Ethiopia.[1]

The language has its alphabet, ፊደል/ fidäl, inherited from the Geez (Ethiopic) language. Geez is an ancient South Semitic language which now serves only as the liturgical language of the Ethiopian Orthodox Tewahedo Church. Fidäl is a syllabary writing system where the consonants and vowels co-exist within each graphic symbol. Unlike majority of its Semitic scripts, such as Arabic and Hebrew, fidäl is written from left to right. The writing system consists of 33 consonants, each having seven 'orders' or shapes depending on the vowel with which a given consonant is combined. The alphabet in the traditional order is given in Appendix C. It is necessary to have a Unicode infrastructure set before one attempts to make use of the fidäl. Because of the syllabary features, due consideration must also be given while dealing with the Amharic strings. There is no standard way to transliterate Amharic into the Latin alphabet.

Having said this much about the fidäl alphabet, let's give a thumbnail summary of the morphology of Amharic words. Nouns in Amharic decline for number (singular/plural), species (definite/indefinite) and case (nominative/accusative/genitive/dative). Unlike these variable features, the nouns exhibit an inherent behavior towards gender, that is, a given Amharic noun is either masculine or feminine. These declensions are usually achieved through affixation. Suffixal affixations are predominant while there are also a good number of prefixings. The above order as well shows the way in which these nominal affixes appear. We show this by giving an example '*the houses*' (definite, plural noun phrase used as a direct object) from the sentence '*I sold the houses.*' and show the Amharic counterpart as follows,

[1] http://en.wikipedia.org/wiki/Amharic_language

the houses-Acc - ቤቶቹን - bet-occ-u-n

Here the suffixes (‘-occ’, ‘-u’ and ‘-n’) are added to the head noun ‘bet’, which means house, to mark declensions for number (Pl), definiteness (Def) and case (Acc) to get the required form – the houses - betoccun (ቤቶቹን).

The pronouns in Amharic can be put into three persons as in English, but there are some unique features such as the second person 'you' which may take different agreements when referring 'plural', 'respected (politeness)','singular-female' or 'singular-male' nouns as shown below.

You (Masc. Sg.)	:	አንተ - antä
You (Fem. Sg.)	:	አንቺ - anči
You (Pl.)	:	እናንተ- ənantä
You (Politeness)	:	እርስዎ- ərswo

Adjectives come before nouns in a sentence to modify them. In a sentence ‘*antä bäTam gobäz tamari näh*’ – ‘you are a very clever student’ there are two adjectives ‘bäTam’ - *very* and ‘gobäz’ – *clever*. ‘bäTam’- *very* modifies ‘gobäz’ - *clever* , and ‘gobäz’ modifies the noun ‘tamari’ – student.

Unlike majority of the languages in the GF library that construct words by linearly concatenating morphemes, Semitic languages have unique non-concatenative properties in addition to the conventional concatenative modifications. Therefore, in Amharic a verb, which is the most complex category of words, is created generally from consonantal radicals which are inflected by a process of merging with vocalic components based on various patterns. The majority of roots, like other Semitic languages, have three radicals. However, there are also a significant number of verbs that are multi-radical and bi-radical. A verb takes various forms depending on the tense-aspect-mood, voice and root structure while inflecting for person gender and number. This is done through the addition of affixes at both ends and even between the roots of a stem.

The numerals in Amharic can assume a cardinal or ordinal form. The ordinals are all the times achieved by adding the prefix *äňňa* on the ordinals.

Cardinal	Ordinal	
ሁለት *hulät / two*	ሁለት+ -ኛኛ -> ሁለተኛ	*hulät-äňňa/ second*

The prepositions appear as simple prepositions that are stand alone or as separate entities coming both at pre and post positions.

To – you	ለ - አንተ - *lä* antä	(pre)
On - you	አንተ - ላይ - antä *lay*	(post)
With – you	ከ - አንተ - ጋር - *ke* antä *gar*	(both pre and post)

The word order in Amharic clauses is generally SOV. Verbs agree with their subjects in number gender and person and objects precede verbs within the verb phrase

In Amharic comparison, both comparative and superlative forms exist nearly following the same trend of formation. The superlative form is like a comparative between the whole and the one in question. This means that the Amharic equivalent of, for example, ‘I am the best’ will be equivalently translated as ‘I am better than the rest’.

There are also other syntactic considerations that need to be made while studying the Amharic grammar. For instance, the definite article in Amharic is a morphologically bound element and its treatment has been a point of discussion for many linguists.

3.2 “*The boy loves this beautiful girl” in GF*

To help us better explain GF's way of grammar engineering; let's take a simple Amharic sentence as an example – 'The boy loves this beautiful girl'. To further clarify our purpose for the wider audience, we translate this to English using our system.

ልጁ	ይህች	ቆንጆ	ልጃገረድን	ይወዳል ::
lə∂ju	*y∂h∂c*	*qonjo*	*lə∂jagäräd∂n*	*y∂wäddal*
boy-DEF	*this-FEM*	*beautiful*	*girl-FEM*	*loves*

The boy loves this beautiful girl.

The parsing tree representation of the above sentence is given in fig.1 below. To give a summary, the purpose of the whole work is to enable the user of the library to obtain Amharic concrete sentences like the above just from descriptions made in the abstract syntax. This abstract syntax, as we tried to note out earlier, is shared by all the languages while the rules governing the concrete syntax vary in accordance with the morphosyntatic properties and lexicon of the language under implementation. This means that each branch of the tree describing some form of syntactic rules is implemented for Amharic. The related morphological rules and lexicon selections together provide the equivalents of the nodes. Let us emphasize more on this using our example.

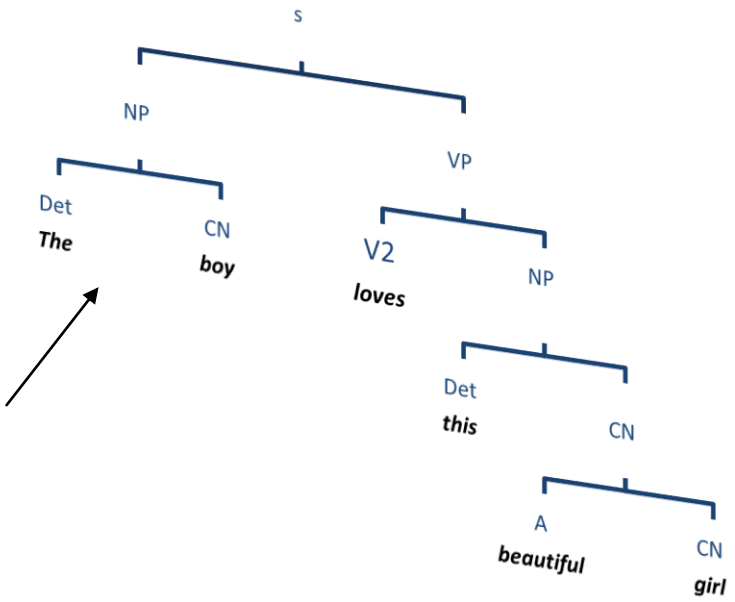


Fig 1: A tree structure representing '*The boy loves this beautiful girl*'

Considering the left branch in the tree which represents the rule in the abstract syntax that takes a determiner and noun to give a noun phrase has the following type signature.

```
fun
        DetCN: Det -> CN -> NP;        (3.1)
```

Where

`DetCN:` the name of the syntax rule

`Det:` the types for the **determiner** and

`CN:` the **noun** component.

For the case of our example Det is -ኡ / -u and the CN is *lɔ̂j* /ልጅ -*boy.* But these separate entities lɔ̂j and -u are combined to form lɔ̂ju / ልጁ- *the boy* showing the definite marker in Amharic is a morphologically bound element. One must be watchful while making such selections as the gender of the determiner which actually comes from the noun. In our example ልጃገረድ/*lɔ̂jagäräd/ -girl* is a feminine noun and this is the reason why ይህች/ *yɔ̂hɔ̂c- this* is selected instead of ይህ/*yɔ̂h* – (masculine counterpart for *this*). In the same fashion the number and definiteness features of the noun come from the determiner while case remains usually nominative.

What we have tried to discuss above is rather a simplified presentation that merely handles a part of the example we picked at the beginning of the section. So as to better understand the generalizations that can be made with these rules and appreciate the way GF deals with such grammatical decisions, we first identify the types of the categories involved and forward the variable and inherent features. The two features are later on connected with the mechanism of agreement where a variable feature of one entity is determined by the inherent feature of the other. For the categories we have in our rule 3.1.

```
lincat

Det = { s : Gender => Case => Str ;
        d : Species;
        n : Number
      } ;

CN = {  s : Number => Species => Case => Str ;
        g : Gender} ;

NP = {  s : Case => Str ;
        n : Number ;
        g : Gender ;
        p : Person
} ;
```

As can be seen the types are put in the form of a GF record
where

`s` is the string

`n` is the number (`Sg` or `Pl`)

`d` is the species that signifies the definitiveness (`Def` or `Indef`)

`g` is the gender (`Masc` or `Fem`)

`p` is the person (`Per1`, `Per2`, or `Per3`)

From the above for instance the type representation for NP states that noun phrases in Amharic have an inherent gender, number and person while they vary in case. The => is the table operator in GF, so having a single input to NP table means that a NP in Amharic is inflected for only this attribute. A complete description of how such multi dimensional table representations work is given in (Ranta, 2010)

However, it is worth noting at this moment that while designing such features we have come to notice a significant amount of similarity with other languages in the library such as Arabic

hinting the framework's capacity of serving as a platform for investigating structural similarities between the languages that are implemented in the library.

Coming back to the story with the boy and the girl, an implementation of the abstract syntax rule in 3.1 can be given for Amharic in the `NounAmh` module as:

```
Lin

DetCN det cn = {
          s = \\ c =>
                 det.s ! cn.g ! c
                ++ cn.s ! det.n! det.d ! Nom ;
          n = det.n;
          g = cn.g;
          p = Per3;

         };
```

The above concrete syntax rule for Amharic has a lot of abstractions in it. First of all the rule accepts a case argument `c` because the NP declines only for case as mentioned above. Of course the roles played by the NP in a given context vary in accordance with the case types. In Amharic these values for case can assume an accusative, nominative, genitive or dative form. The rule goes on to state that the string of the determiner is directly brought from its own inflection table once we have the gender of the noun and case of the noun phrase. Extending the same pattern, the string of the noun is selected by making use of the number and definiteness of the determiner while the case remains nominative.

After we choose those Det and CN strings, we concatenate them using the concatenation operator ++ to form the NP's string. It would be an error to leave this operator out and use a simple juxtaposition as GF is a functional programming language and juxtaposition is used as a notation for function application. Having done the string part further consideration is given for agreement features where the noun phrase receives its number from the determiner and its gender from the noun. This is how the elements formally complete each other to support what we descriptively explained about ይ-ህች ቆንጆ ልጃገረድ-ን / *y∂h∂c qonjo l∂jagäräd∂n* at the beginning of the section.

We have been so far discussing the abstract morphosyntatic features of the API and how they work underneath through an example. This will, of course, not be complete without the lexical entries of normal and structural words that the user requires. This means that the Amharic application programmer expects to get the necessary word entries. If this is not available, taking the case of our example, entries can be made a follows

```
boy_N = mkN "ልጅ";
```

```
girl_N = mkN "ልጃገረድ" feminine;
beautiful_A = mkA "ቆንጆ";
love_V2 = mkV3gdl "wdd";
```

Where `mkN`, `mkA` and `mkV3gdl` respectively specify the lexical paradigms for regular nouns, the lexical paradigms for regular adjectives, and triradical verb of the *gdl* (ገደለ) family

The `feminine` entry after the second `mkN` signifies the fact that ልጃገረድ/l ə jagäräd - *girl* is a feminine noun. There are very few nouns that follow this paradigm while the default gender of a noun is taken to be masculine. Given, for instance the three roots '*wdd*, our system is capable of generating most Amharic stems with minimal error in inflection and orthography.

In this chapter we have tried to give a brief introduction to the Amharic language and the GF formalism we have used to implement it and how it works. We also picked an example and operated on it to see how it can be generated in GF and explained the works that happen behind the scenes while doing so. The next chapter emphasizes more the overview of the system by describing the modules and components it is made of. We do so to lay the ground work for the implementation we show in chapter 5.

Chapter 4

System Overview

Having justified the importance of such a work in the Amharic NLP, discussing the important features of the language and showing how these features are implemented in the GF formalism, we now lay the grounds for our work by giving an overview the modules and components from which our system is made of. It is also our intention to sometimes give the reader only the references to the grammar files (*.gf) where the details are found rather than discuss the whole code in this report.

4.1 Grammar Files

GF adopts a module system which is useful for software engineering and separate compilation. The two major GF modules are the abstract and concrete syntax modules. These two are used at run time while parsing and generating. They are arranged into hierarchies in similar fashion as object oriented programs whereby grammar sharing is enabled at top-level grammars. Resource module is a channel module for sharing code across other concrete modules independently. Here, we rather give the system overview of the grammar files that we covered ranging from such linguistic features as the morphology, syntax and lexicon. We also introduce the transliteration technique adopted to handle the need for Unicode infrastructure by Amharic orthography.

4.1.1 Orthography

Amharic has its own syllabic alphabet that requires close orthographic treatment when forming words. For example, the operations that must be considered while inflecting for definiteness and number, palatalization rules where dental sounds get changed to plate sounds, and the various conjugations of verbs all need due orthographic consideration when they are written down.

4.1.2 Morphology

Morphology in a resource encompasses the sets of operators that help transform one entity into another with the required morph or form. For instance, the resource morphology of Amharic consists of verb conjugations that are expressed as operation and paradigms used to handle the declensions of nouns and adjectives. This includes different files in the modules that cover parts

of the complex Amharic morphology. We first present the types that are needed in the resource module `ResAmh`, thereby discussing the roots and patterns feature and how they help in verbal morphology. We take into consideration radicals that range from two to four to show our purpose. In `PatternsAmh` we try to give the major patterns that the Amharic verbs take. We do not dare to claim that we have covered every possible pattern of the language, rather the majority of bi, tri and quadriradical verbs can be formed using the patterns. In addition to the definition of tables for the nominal morphologies in the `ResAmh` module, a set of paradigms which are used while building the lexicon are given in the `ParadigmsAmh` module. We did this to increase the loading efficiency during compilation.

4.1.3 Syntax

The implementation of the basic syntax rules is distributed across various modules complying with the requirements of the resource grammar API. These modules include the various phrase categories that describe the phrase construction in that specific category such as noun, verb, adjective, sentence, etc.

4.1.4 Resource Lexicon

A resource lexicon is a set of words paired with paradigms. In the system there are two independent lexical modules: the `StructuralAmh` and the LexiconAmh. The first contains sets of structural words such as (determiners, prepositions, etc) while the later is the main module where a list of few hundred words (nouns, verbs, adjectives etc) from the Swadesh list – which is a list of basic lexical terms compiled by Morris Swadesh[1], is included. It is a general trend that lexica of a language keep evolving all the time, either with the addition of newer words or inflection of older words with newer paradigms; hence, the inflection of these words only represents the features of the present day Amharic.

4.2 Transliterations

Strings in the GF grammar files are built from Unicode characters internally. This makes it easy to handle languages such as Amharic which use their own set of writing system. We had difficulty in writing Amharic characters in iso-latin-1 as the latest of everything with Amharic keyboards is Unicode, plus trying to save the grammar file in any other format but Unicode or UTF-8 creates series of '?' instead of the Amharic characters we wanted to see. So we have defined our own non-ASCII character sets through transliteration. Amharic characters are

[1] The Swadesh list is one of several lists of vocabulary with basic meanings, developed by Morris Swadesh. http://en.wikipedia.org/wiki/Swadesh_list

represented in the range from 1200 to 137F in the Unicode range. This makes the language the largest set in the world with 384 unique characters, including the reserved codes. Fidäl contains some characters which are identical and thus could be used interchangeably in modern Amharic. In old days Amharic, as in the case of Geez, the cases of using these similar sounding letters was given due attention but we do not consider all of that here. We make the decision not based on any linguistic fact but for the mere sake of brevity. This leaves the transliterated table with only 344 characters as shall be seen in the appendix section. The rules that we followed while defining our transliteration table were already predefined so we had to stick to them. These rules stated that: the transliteration should be a letter or a letter character and non letter character, in addition capital and small letters should be treated separately. These have restricted us from using an already existing, but not officially standard, phonetic transliteration scheme for Amharic known as SERA[1].We have given in appendix A the transliterations for our 344 characters that span from 0x1200 to 0x1357 in the Unicode range .

[1] http://www.geez.org/IM/

Chapter 5

Implementation of Amharic in GF

In the previous chapters of this report we have been trying to lay the foundation of our work by describing the Amharic language through examples and introducing the major components that our system is made of. We now start with the implementation of these components in the order that they appear in the specification.

5.1 Orthography

The fidäl[1], Amharic's alphabet, is written from left to right and is composed of thirty three consonants and seven vowels. These are arranged into seven houses (orders) according to the kind of each vowel that the consonants associate themselves with, i.e., the consonant-vowel (CV) combinations. Consider for instance the following individual symbols:

ቨ	ቩ	ቢ	ባ	ቤ	ብ	ቦ
bä	bu	bi	ba	be	bə	bo

ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ
tä	tu	ti	ta	te	tə	to

However, if a string begins with a vowel then the vowel is written independently; which means, there are no symbols added to the individual vowels.

ኧ	ኡ	ኢ	ኣ	ኤ	እ	ኦ
ä	u	i	a	e	ə	o

In such cases as affixation, a vowel may come in contact with a consonant on its left side. During such cases the vowel will no longer be considered independently but together with the immediate preceding consonant forming a new unique symbol. We can better clarify this by

[1] A complete list of the Amharic fidäl is given in Appendix C

taking a case for the declension of a noun for number by adding *-occ* / -ኦች at the end of the noun. As can be seen in the example below the combination of ት and ኦ results in ቶ.

ቤት + -ኦች = ቤትኦች
ቤቶች
bet + -occ = *betocc*
house houses

Below we give the implementation of such changes by defining an operation that takes a word as an input and gives an orthographically sound word. The GF's regular expression is such a handy method to do this as can be seen in `replaceLastLet6_7`. The operation helps to change the sixth orders to the seventh while declining for number, and hence the nomenclature. The key word `oper`, that denotes operation, is placed at the beginning to convey the fact that `replaceLastLet6_7` is a function.

```
oper
replaceLastLet6_7 :  Str -> Str = word ->
                  let y = last word in
                           case y of {
                                                "ህ" =>"ሆች" ;
                                                "ል" => "ሎች" ;
                                                "ሕ" => "ሖች" ;
                                                "ም" => "ሞች" ;
                                                "ሥ" => "ሦች" ;
                                                      ...

                                                "ት" => "ቶች";
                                                      ...
                                                _ => word+"ዎች"
                                                  } ;
```

In addition to such cases which happen during pluralization, similar trends of transformation may occur in the declension for definiteness. For instance, if the noun in question ends in the sixth order and its gender is masculine, then it only changes this ending letter to the second order during the declension for species (definiteness). Otherwise, it always takes *-wu*/ ዉ at the end. Similarly for feminine nouns the same trend of order change holds but they instead add *-wa*/ ዋ at the end and so on.

5.2 Verbal Morphology

5.2.1 Introduction to Non-Concatenative Morphology

Unlike majority of the languages in the GF library that construct words by linearly concatenating morphemes, Semitic languages have unique non-concatenative properties in addition to the

conventional concatenative modifications. Therefore in Amharic, verbs are created mostly from consonantal radicals or *roots* which are inflected by a process of merging with vocalic components based on the various *patterns* which are sequences of vowels and consonants into which root consonants are inserted. In Amharic, even if majority of the verbs are triradical (three consonantal roots), the number of these radicals may range from two to five and more. In our implementation we have only considered those between two and four radicals. We define these by using record types that help to represent words as data structure as follows:

```
Root2 : Type = {C1,C2: Str};
Root3 : Type = Root2 ** {C3 : Str};
Root4 : Type = Root3 ** {C4 : Str};

Pattern2 : Type = {C1,C1C2,C2: Str};
Pattern3 : Type = {C1,C1C2,C2C3,C3 : Str};
Pattern4 : Type = {C1,C1C2,C2C3,C3C4,C4 : Str};
```

where `C1, C2 ,C3` and `C4` stand for the consonantal radicals while, for instance in Pattern2 `C1,C1C2` and `C3` represent the head, the middle and the tail of the pattern. The `**` operator in GF is used to add an additional field into a record and hence, `Root2` is biradical while `Root3` has an additional one consonant making it a triradical and similarly `Root4` is quadriradical.

The root itself has no definite pronunciation until combined with the appropriate pattern. Such combinations are non-linear making them rely on two independent root and pattern. In Amharic there are different ways in how templates modify the root consonants: doubling the middle consonants, inserting vowels between consonants, adding consonantal affixes, etc.

We do such interwinings of roots and patterns by implementing two functions that help extracting each root from a given root-string and applying a selected pattern on the extracted roots to make the final form. We show how these two work flows are applied by considering triradicals :

```
getRoot3 : Str -> Root3 = \s -> case s of {

 C1@? + C2@? + C3 => {C1 = C1 ; C2 = C2 ; C3 = C3} ;

                  _ => Predef.error ("cannot get root from" ++ s)

                       };


appPattern3 : Root3 -> Pattern3 -> Str = \r,p ->

    p.C1 + r.C1 + p.C1C2 + r.C2 + p.C2C3 + r.C3 + p.C3 ;
```

The operation `getRoot3` associates every consonant in the input string `Str` with a variable. This is achieved by using the operation `C@` which binds each consonant in the strings to a variable, e.g. `C1,C2` and `C3`. These variables are then coded into patterns using the operation `appPattern3` which specifies how the root's consonants should be inserted into a pattern, i.e., given a root and a pattern. The final output is just a concatenation of the seven strings , without dropping any. We can take example patterns to demonstrate our purpose. These patterns specify the consonant slots and morphological forms as shown below

```
C1aC2aC3a =   {C1 = "" ; C1C2="ä"; C2C3 ="ä"; C3="ä"};

meC1C2aC3 =    {C1 ="mä"; C1C2="" ; C2C3 ="ä"; C3="" };
```

For example, when the root *sbr* is applied to the first pattern `C1aC2aC3a`, it forms the perfect stem ሰበረ /*säbärä* (he broke) while when applied to the second pattern it forms an Amharic infinitive ማስበር/ *mäsbär* (to break). Of course these examples do not show gemination as the Fidäl alphabet does not indicate the same and therefore our Amharic outputs do not provide this. However such properties play a significant role while classifying the verb conjugations. In addition, marking gemination in some format is of course useful when developing speech applications using the language and we leave that as a future work. We now proceed to our survey of Amharic verbs and later on show how these root-pattern combinations are implemented for their morphology.

5.2.2 Survey of the Amharic Verb

As we stated earlier, Amharic verb forms are derived by applying various templates (vowel and affix patterns) to a set of roots consisting of between two to five consonants. The prefixes and suffixes are all grammatical morphemes that are added to the stem while the stem remains the lexical part of the verb and also the source of most of its complexity. Verbs in Amharic are marked for person, number, and gender and the different forms can be described as varying along the following grammatical dimensions:

Tense/Aspect/Mood

Tense/aspect/mood (TAM) is signaled mainly by the prefixes and suffixes that indicate the subject of the verb but is also reflected in the stem template.

Traditionally the four main TAM's: perfective (or perfect), imperfective (or imperfect), jussive /imperative (the imperative is just another form of the jussive when applied on the second-person), and gerundive (or gerund) are usually given by scholars. We have extended this with the four 'extra' forms: infinitive, participle, compound-perfect and the contingent that form verbs with auxiliaries. Appendix B shows the various verb forms.

Here are non-geminated examples of each of the main TAM's when considered for the third person singular masculine (`Per3 Sg Masc`), active (`Act`) voice. The root of the verb is *sbr* – to break.

- Perfective: ሠበረ *säbärä* / he broke
- Imperfective: ይሠብራል *yə-säbr-al* / he breaks /he will break
- Jussive: ይሥበር *yə-sbär* / let him break
- Imperative: ሥበር *sbär* / break!
- Gerundive: ሠብሮ *säbr-o*/ having broken

Voice

For the verb forms we considered above, the voice is signaled by the stem prefixes *tä* as well as particular patterns of vowels between the root consonants. The first example below shows the first case by taking *tä* as a prefix on the stem *säbärä (to break)* while the second one takes '*ä*' in between the second and third roots (bear in mind that *sbr* is the root form).

Passive: ተሰበረ **tä**-säbärä (he is broken) ይሰበራል yə-s*äbä*r-al (he will be broken)

The stem-internal aspect does also give yet another dimension of the templates in addition to the values of the TAM and voice features. This can assume the normal or simple form and two other forms (reciprocal and internal). We only take the normal form in our implementation as almost all of the API rules ask for the same.

Stems: roots

The majority of roots in Amharic, like other Semitic languages, have three radicals. However, there are also a significant number of verbs that are multiradical and biradical.

Example with two, three, four, and five radicals is given below:

1. ሰማ *säma* 'he heard',
2. ገደለ *gädälä* 'he killed',
3. መሰከረ *mäsäkärä* 'he testified' and
4. ተብረከረከ (tä)*bräkäräkä*'he got trembled'

Considering the vocalic structure and the germination or non germination of the 2nd radical in the triradical, the verb has three types: types A, B and C. Regardless of the type , the 2nd radical is always geminated in the perfect. The reader of this report should note that we have come up with the derivational root patterns and only briefly adopted the verb classification from previous scholars to fit our purpose.(Leslau 1969). We now describe these commonly occurring root patterns of the bi, tri and quadri radicals.

BIRADICALs

In the biradical, the 2nd consonant is geminated in the perfect only if it is the 2nd radical of the root.

Class mkV2bl
Type A: mkV2bl በላ/ *bälla* ‘eat’
Type B: mkV2TT ጠጣ /*tätta* ‘drink’
Type C: mkV2qT ቀጣ /*qatta* ‘punish’

Class mkV2sT
Type A: mkV2sT ሰጠ /*säTTä* ‘give’
Type B: mkV2ly ለየ /*läyyä* ‘distinguish’
Type C: mkV2wN ዋኘ /*waNNä* ‘swim’.

The other biradicals considered in the implementation have only one type. These classes are
Class mkV2yz ያዘ /*yazä* ‘hold/capture’
Class mkV2nr ኖረ /*norä* ‘live’
Class mkV2hd ሄደ /*hedä* ‘go’

A special verb አለ / *alä* ‘say’ is also implemented independently.

TRIRADICALs
mkV3gdl: ገደለ *gäddälä*, characterized by lack of germination of the 2nd radical in the verb forms other than the perfect.
mkV3mls: መለሰ, *mälläsä*, characterized by the germination of the 2nd radical in all the verb forms.
mkV3brk: ባረከ *barräkä*, characterized by the vowel a after the 1st radical for the germination or non germination of the 2nd radical. mkV3tTb : ታጠበ *tattäbä* is yet another variation of such triradical classes having differences in the imperative/ jussive forms.

The above first three forms for triradicals mkV3gdl, mkV3mls and mkV3brk from now on called TYPEA, TYPEB and TYPEC respectively, do usually lay foundation work for the variation of the rest of the other classes. In other words, it is usual that the others (including bi, tri, ...) can inherit these basic forms and then add their unique features.

The other triradical verbal classes considered include those that have initial labiavelerals ኦ /'o' or ኡ/ 'u' after the initial radical.

mkV3qTl: ቆጠረ *qoTTärä*, count
mkV3qfr: ቆፈረ *qoffärä*, dig
mkVqTr: ቋጠረ *quaTTärä*, tie

A special class for the triradical is one that accounts for the verbs beginning with አ /a, which is a vowel itself. This puts extra challenges in the orthographic and phonetic analysis of the gemination process.

mkV3asr: አሰረ *assärä* ‘pass’ which
mkV3asb: አሰበ *assäbä* ‘think’

QUADRIRADICALs

In the quadriradical, the 3rd radical is geminated in the perfect. The quadriradical have two types:
mkV4dbdb: ደበደበ / *däbäddäbä* ‘kick’
mkV4qlql 2: ቀላቀለ/ *qälaqqälä* ‘mix’ characterized by the vowel a after 2nd radical. The gemination or the non gemination of the 3rd radical in the verb form other than the perfect is the same in both types.

5.2.3 Implementation of Verbal Morphology

In section 5.2.1 we have shown that, for example, typical triradical root operations include, pattern - which is a string consisting of a four position pattern slots and root - which is a string consisting of three consonant roots. Furthermore we have given the analysis of a triradical root showing the mechanism for achieving this non- concatenative inflection.
In the resource grammar, the verb is represented as

```
param
          Number = Sg | Pl;
          Gender = Masc | Fem;
          Voice  = Act|Pas;
          VForm = Perf|Imperf |Jus_Imperat|Gerund|Infinitive|
          Parti|CompPerf|Cont;
          PerNumGen = Per1 Number
                    | Per2 Number Gender
                    | Per3 Number Gender;

oper

          Verb = {s: VForm =>Voice=> PerNumGen => Str }
```

The first parameter `VForm` details the TAM constructors of the verb forms, while the additional parameter `PerNumGen` provides a detailed description of how verbs are inflected with regard to person, number and gender. The three constructors of the `PerNumGen` indicate: first person singular/plural, second and third person singular/plural and masculine/feminine showing that Verb lexemes are inflected for person, number, gender, voice and form. Parameter types are similar to algebraic data types in functional programming languages. Such a representation avoids considering the cross products of the atomic members like person, number and gender. For instance, `PerNumGen` gives a more compact representation (2+4+4 =10) than listing all the constructors (3x2x2 = 12). The above three dimensions (Mood, Aspect (TAM)-(8) and Voice-(2)) when applied to the 10 possible combinations of person-number-gender give a crude table

with 160 forms. We tagged them as being 'crude' because such forms as 'participles' and 'infinitives' show the same trend of variation for every person, gender and number and thus should be counted as unit.

Following the discussion and example in section 5.2.1 a triradical verb-inflecting-operation such as mkV3gdl defines regular verb paradigms for each form and agreement features shown above as follows :

```
mkV3gdl : Str -> Verb = \v ->
                    let root = getRoot3 v
                    in {
                    s = table {
          Perf => table {
               Per1 Sg      => appPattern3 root C1aC2aC3ku ;
               Per1 Pl      => appPattern3 root C1aC2aC3n ;
               Per2 Sg Masc => appPattern3 root C1aC2aC3k ;
               Per2 Sg Fem  => appPattern3 root C1aC2aC3sh ;
               Per2 Pl _    => appPattern3 root C1aC2aC3achehu ;
                                        ...
                                            };
          Imperf => table { ... }
          ...
  }
};
```

The original patterns that signify the consonant slots and morphological forms have a transliterated version of the vocalic patterns as described in section TO DO:

```
          C1aC2aC3ku = { C1 ="" ; C1C2="'"; C2C3 ="'"; C3="k&"};
          C1aC2aC3n  = { C1 = "" ; C1C2="'"; C2C3 ="'"; C3="n"};
          C1aC2aC3k  = { C1 = "" ; C1C2="'"; C2C3 ="'"; C3="k"};
          C1aC2aC3sh = { C1 = "" ; C1C2="'"; C2C3 ="'"; C3="x"};
                      ...
```

Here below we summarize the results of how the parameters mentioned can be used . This very much attests to the fact the GF formalism is indeed carefully engineered to parallel the way grammarians think of languages. This is shown by taking the root form *'sbr'* – to break and inflecting it with the conjugating operation mkV3gdl described above (i.e. by executing mkV3gdl against "*sbr*"). We only show the perfective verb forms in an active voice while the details of the rest can be seen in Appendix B:

`Perf Act (Per1 Sg)` : ሠበርኩ *säbbärku – I broke*
`Perf Act (Per1 Pl)` : ሠበርን *säbbärn – We broke*
`Perf Act (Per2 Sg Masc)`: ሠበርክ *säbbärk – You broke (for Masc)*
`Perf Act (Per2 Sg Fem)` : ሠበርሽ *säbbärx –You brike (for Fem)*
`Perf Act (Per2 Pl Masc)`: ሠበራችሁ *säbbärachu Your broke (for plurals of any gender)*–same for : `s Perf- Act (Per2 Pl Fem)`

```
Perf Act (Per3 Sg Masc): ሠበረ  säbbärä – He broke
Perf Act (Per3 Sg Fem) : ሠበረች säbbäräc – She broke
Perf Act (Per3 Pl Masc): ሠበሩ säbbäru – They broke – same for : Perf Act (Per3 Pl Fem)
...
```

It can be noted from here that the perfect form is made by inserting vocalic elements and adding suffixes to identify the Person Number and Gender combinations except in cases of second and third person plural that remain the same for both genders. Such situations in GF, like other functional programming languages such as Haskell, are handled by a wild card patterns '_'. The rest of the verb forms can be handled in similar fashions as variations in the type of vocalic patterns, the position of the vocalic pattern's insertion, the kinds of affixes that can be added and so on.

The transliterated intermediate form of the above result looks like this :[1]

```
Perf Act (Per1 Sg) :        s'b'rk&
Perf Act (Per1 Pl) :        s'b'rn
Perf Act (Per2 Sg Masc) :   s'b'rk
Perf Act (Per2 Sg Fem) :    s'b'rx
Perf Act (Per2 Pl Masc) :   s'b'r!ch&
Perf Act (Per2 Pl Fem) :    s'b'r!ch&
Perf Act (Per3 Sg Masc) :   s'b'r'
Perf Act (Per3 Sg Fem) :    s'b'r'c
Perf Act (Per3 Pl Masc) :   s'b'r&
Perf Act (Per3 Pl Fem) :    s'b'r&
Perf Pas (Per1 Sg) :        t's'b'rk&
Perf Pas (Per1 Pl) :        t's'b'rn
...
```

We note from this transliterated result that an outstanding achievement can be found by treating the inflection this way as the vocalic patterns can easily get concatenated with the roots and the inflected stem is displayed back into Amharic. One such orthographic error could happen when the final vowel of the Singular - Second Person – Feminine verb is of the third order (ends with '-i'). This brings about the palatalization of any dental, or sibilant where the vowel 'i' is absorbed by the palatal sound. This happens while making such verb forms as imperative, imperfect and participle. So as to handle such a phenomenon we define an operation that can manage to change the graphic symbols whenever they happen as follows:

```
oper
        pallatalize :  Str -> Str = \c->
```

[1] The reader can refer to Appendix A for a transliteration table.

```
case c of {
          "d" => "j";
          "t" => "c";
          "T" => "C";
          "n" => "N";
          "l" => "y";
          "s" => "x";
          "z" => "Z";
          "S" => "C";
           _  => c
           } ;
```

This is then implemented in the following operation to bring the effect that we want by using it instead of `appPattern3` whenever appropriate.

```
appPattern3pal : Root3 -> Pattern3 -> Str = \r,p ->
p.C1 + r.C1 + p.C1C2 + r.C2 + p.C2C3 + pallatalize (r.C3) +
p.C3;
```

Below we give two imperative examples for a third person female to show results before and after we introduced palatalization.

Before correction	After Correction
`gdl - kill (Fem)`	`gdl- kill (Fem)`
`gd'l#`	`gd'y#`
ግደሊ (?)	ግደይ
`lbs - dress (Fem)`	`lbs- dress (Fem)`
`lb's#`	`lb'x#`
ልበሢ (?)	ልበሽ

This template based approach has even given us the possibility of handling verb roots like አዳነ *adn* – to save that start with a vowel አ /a easily, we first get rid of the vowel 'root' (i.e leaving `'r.C1'`) to get *'dn* which we treat as a normal triradical. The vowel is then added in place of the removed root.

```
appPatternRemove : Root3 -> Pattern3 -> Str = \r,p ->
p.C1 + a + p.C1C2 + r.C2 + p.C2C3 + r.C3 + p.C3;
```

5.3 Morphology of Nouns

In our implementation of the Amharic resource grammar, the noun is represented as follows:
`Param`

```
Number        = Sg  | Pl;
Species       = Def | Indef;
Case          = Nom | Acc | Gen| Dat;
```

```
    Gender          = Masc| Fem;

oper

Noun : Type = {    s : Number => Species => Case => Str;
                   g : Gender
               } ;
```

The GF syntax of the noun type and parameters it depends on follow the same trend as the verbs in section 5.2.3 but here it states that nouns (N) inflect in number (singular or plural), species (definite or indefinite) and case (Nominative, Accusative, Genitive or Dative) while a given noun has an inherent gender, that is, no noun is both feminine and masculine at the same time. This record representation of the noun has an s field as multidimensional table storing the (2 x 2 x 4 = 16) forms of the noun.

5.3.1 Number of the Noun

In Amharic, the nouns have both the singular and the plural forms. The plural suffix added on all nouns is either /-አች/-occ or /-ዎች/-wocc, the former for nouns ending in a consonant and the latter for nouns ending in a vowel. There are some additional ways of inflecting some nouns for number, especially those inherited from Geez, but they will not be considered here as equivalent forms can easily be formed this way.

5.3.2 Species / Definiteness of the Noun

The Amharic definite article is a suffixed element and has different realizations depending on whether the noun to which it is attached ends in a consonant or a vowel, singular or plural, and masculine or feminine. If the noun to which it is attached is masculine singular and ends in a consonant, the suffix added to mark definiteness is /ኡ/- *u* as in /ቤት/ - *bet* ‘house’, which becomes /ቤት-ኡ = ቤቱ/ *bet-u* ‘the house’. If the masculine singular ends in a vowel, the definite suffix will be /-ው/-w as in /ሬሳ/ - *resa* ‘corpse’, which becomes /ሬሳው/*resa*-w ‘the corpse’. It is worth mentioning the fact that this definite article homonyms with the possessive marker for third person masculine singular that is *bet-u* can also mean 'his house' and so on.

On the other hand, if the noun to which the definite suffix is attached is feminine singular and ends in a consonant, the marker is realized as /-ዋ/-wa, /-ኢቱ/-itu, or /-ኢትዋ/-itwa (twa) used interchangeably as in ልጂት-*lǝjǝt* ‘girl’ which becomes *lǝjǝtwa*/ *lǝjǝtu/ lǝjtitu* ‘the maid’. If the noun is feminine singular and ends in a vowel, the suffixed element is /-ዋ/ -*wa*, /-ይቱ/ - *yitu* (-*ytu*) or /-ይትዋ/-*ytwa*, again used interchangeably, as in ዶሮ- *doro* ‘hen’, which become *dorowa*/ *doroy-tu*/ *doro- ytwa* all meaning ‘the hen’. In this library implementation the first of the three is adopted as it is common in present day spoken Amharic and it avoids considering the vowelness/consonantness of the last leter of the nound in question making it less laborious to implement.

The definite suffix added to plural nouns ,regardless of gender of the noun, is -u. For example, in the masculine, as in /ንጉሶች/ *nə∂gusocc* ‘kings’ (the plural of *nə∂gus* ‘king’), the definite form becomes /ንጉሶቹ/*nə∂gusocc-u* ‘*the kings’*. In the feminine, as in /ንግስቶች/*nə∂gə∂stocc* ‘queens’ (the plural of *nə∂gə∂st* ‘queen’), the definite form becomes /ንግስት -ኦች - ኡ = ንግስቶቹ/ *nə∂gə∂stocc-u* ‘the queens’.

5.3.3 Gender of the Noun

Some nouns take a feminine marker *-it /-ኢት*: *l∂j* /ልጅ /'child, boy' vs. *l∂j-it* /ልጅ- ኢት = ልጂት/’girl’. But there are ample other masculine nouns that end with *-it /-ኢት*. The feminine gender is not only used to indicate biological gender, but may also be used to express diminutiveness, e.g. *bet-it-u* 'the little house' (lit. house-FEM-DEF). The feminine marker can also serve to admire beauty or express sympathy. This makes it almost impossible to automatically infer the gender of a noun from its structure and construct a smart paradigm function that could infer the gender of the noun from the last letter of the singular form. Majority of the nouns in Amharic take the masculine gender. In the GF Lexicon for Amharic there are more than 175 nouns considered of which only less than 10% are inherently feminine attesting to this fact.

5.3.4 Cases of the noun

The declension of nouns is very simple and uniform. Nouns are inflected through four cases, equally in the singular and the plural,i.e., the nominative, the genitive, dative and accusative.

One example may suffice to show the while mode of proceeding

	Singular	Plural
Nom:	ቤት-bEt - a house	ቤቶች /bEt-occ - houses
Gen:	የቤት-y*ä*-bEt - of a house, a house's	የቤቶች/*yä*-bEt-occ - of houses
Dat:	ለቤት-*lä*-bEt - to a house	ለቤቶች/*lä*-bEt-occ - to houses
Acc:	ቤትን- bEt-n- a house	ቤቶችን/bEt-occ-n - houses

As can be seen from above the nominative is characterized by the total absence of outward indicators. It can also be generalized that accusative endings comes after all other noun formats i.e. at first is added the number indicator then comes the accusative case. The meanings of the noun's genitive and dative cases are conveyed by the possessive prefixes -የ /-y*ä* : and -ለ /-l*ä* respectively.

The following operation defines the affixes that attach to strings to inflect them for case. Given an input string the above declensions for case can clearly be coded as shown below.

```
oper affix : Str -> Case => Str = \str->
        table {
                  Gen => "የ" + str;
                  Dat => "ለ" + str;
                  Acc => str + "ን" ;
```

```
                    Nom => str
                };
```

When it comes to accounting for the above inflections we define an inflection table that internally represents all the sixteen cases we outlined at the beginning of this section.

```
mkNoun : (x1,_,_,_,_,_,_,_,_,_,_,_,_,_,_,x16 : Str) -> Gender -> Noun =

\sdn,sda,sdg,sdd,sin,sia,sig,sid,pdn,pda,pdg,pdd,pin,pia,pig,pid,g -> {
            s = table {
                  Sg => table {
                                  Def => table
                                            {
                                              Nom => sdn ;
                                              Acc => sda ;
                                              Gen => sdg ;
                                              Dat => sdd
                                            };
                                  Indef => table
                                            {
                                              Nom => sin ;
                                              Acc => sia ;
                                              Gen => sig ;
                                              Dat => sid
                                            }
                                } ;
                  Pl  => table {
                                  Def=> table
                                            {
                                              Nom => pdn ;
                                              Acc => pda ;
                                              Gen => pdg ;
                                              Dat => pdd
                                            };
                                  Indef => table
                                            {
                                              Nom => pin ;
                                              Acc => pia ;
                                              Gen => pig ;
                                              Dat => pid
                                            }
                                }
            } ;  g = g
                        } ;
```

The table serves as a template to form other paradigms that account for the 16 forms (`sdn or (Sg,Def,Nom), sda or (Sg,Def,Acc), etc`). The details of orthographic changes we discussed in section 5.1 are now used here. In addition it gives the possibility of handling cases of the inherent features `g` as shown below.

```
            regN2 :  Str -> Gender -> Noun = \root,g ->
                                    case root of {
```

```
                roo + t@? => table {

        Masc => mkNoun
                (roo + replaceLastLet6_2_M (t))
                (roo + replaceLastLet6_2_M (t)+"ን")
                . . .
                g;

        Fem  => mkNoun
                (roo + replaceLastLet6_2_F (t))
                (roo + replaceLastLet6_2_F (t)+"ን")
                . . .
                g
        } ! g
                };
```

Depending on the gender of the noun in question, analysis is made on its last character by making use of the othographic operations for the 16 cases discussed earlier. Below is shown an example obtained from such analysis we made for the Amharic noun for 'house' *bet/*ቤት

```
Sg Def Nom          : ቤቱ
Sg Def Acc          : ቤቱን
Sg Def Gen          : የቤቱ
Sg Def Dat          : ለቤቱ
Sg Indef Nom        : ቤት
Sg Indef Acc        : ቤትን
Sg Indef Gen        : የቤት
Sg Indef Dat        : ለቤት
Pl Def Nom          : ቤቶቹ
Pl Def Acc          : ቤቶቹን
Pl Def Gen          : የቤቶቹ
Pl Def Dat          : ለቤቶቹ
Pl Indef Nom        : ቤቶች
Pl Indef Acc        : ቤቶችን
Pl Indef Gen        : የቤቶች
Pl Indef Dat        : ለቤቶች
```

Before concluding our report on the works done with Amharic nouns, let's mention one special case of compound nouns. In Amharic two nouns may combine to form another noun such as ትምህርት ቤት - *təməhərət bet* 'school' which is formed by two nouns ትምህርት- *təməhərət*- 'school' and ቤት - *bet '* house'. Here inflection for a dative or genitive case is added on the first element while accusative case and the plural markers are suffixed on the second noun. compN is special function that can handle such variations as shown below. It takes two nouns and forms a compound noun with the correct inflections.

```
compN : Noun -> Noun -> Noun ;
```

```
compN x y =
  {
    s = \\N,S,C => case C of
         {
```

```
Gen|Dat => x.s ! Sg ! Indef!C    ++ y.s ! N ! S ! Nom ;

        _  => x.s ! Sg ! Indef!Nom  ++ y.s ! N ! S !C
             };

               g = y.g;

              } ;
```

A GF output for such an analysis can be shown here for the noun school **ትምህርት ቤት** - *təməhət bet* 'school'

```
Sg Def Nom          : ትምህርት ቤቱ
Sg Def Acc          : ትምህርት ቤቱን
Sg Def Gen          : የትምህርት ቤቱ
Sg Def Dat          : ለትምህርት ቤቱ
Sg Indef Nom        : ትምህርት ቤት
Sg Indef Acc        : ትምህርት ቤትን
Sg Indef Gen        : የትምህርት ቤት
Sg Indef Dat        : ለትምህርት ቤት
Pl Def Nom          : ትምህርት ቤቶቹ
Pl Def Acc          : ትምህርት ቤቶቹን
Pl Def Gen          : የትምህርት ቤቶቹ
Pl Def Dat          : ለትምህርት ቤቶቹ
Pl Indef Nom        : ትምህርት ቤቶች
Pl Indef Acc        : ትምህርት ቤቶችን
Pl Indef Gen        : የትምህርት ቤቶች
Pl Indef Dat        : ለትምህርት ቤቶች
```

5.4 Morphology of the Adjectives

As we mentioned in chapter three of this report adjectives in Amharic precede the nouns that they modify.

እርሱ ሰነፍ ተማሪ ነው ።
ərsu sänäf tämari näw
He lazy student is

"He is a lazy student."

In this example, the adjective *sänäf* "lazy" precedes the noun *tämari* "student" which it modifies.
Adjectives (A) are represented in our resource grammar as

```
A = {s : Gender => Number => Species => Case => Str} ;
```

This builds the representation table of an adjectives, consisting of 32 (2x2x2x4) forms. There is a great deal of similarity between nouns and adjectives when it comes to inflection except that the gender is no longer treated as an inherent feature in the adjectives. This means that one form of adjective can be used to describe both masculine and feminine nouns in some of the cases.

This representation table is also so uniform among adjectives that a single paradigm `mkA` is used to build the whole representation of the more than 50 regular adjectives in our Swadesh Lexicon list. A lexicographer can, for example, enter a new adjective in this list as shown below:

`beautiful_A = mkA "ቆንጆ" ;` where `mkA` is an operation that transfers a given string to an adjective table (`mkA : Str -> A`) inflecting it along the way to give a tabular output such as:

```
Masc Sg Def Nom          : ቆንጆዉ
Masc Sg Def Acc          : ቆንጆዉን
Masc Pl Def Acc          : ቆንጆዎቹን
Masc Pl Indef Nom        : ቆንጆዎች
Fem Sg Def Dat           : ለቆንጆዋ
...
```

5.4 The Numerals

Now we forward the explanation of our implementation of the numeral system of Amharic. We consider both the cardinals and ordinals while doing so. Our work on the numerals is basically an extension of a previous work for defining number systems of various languages using GF. We gradually give the governing grammar rules along with their formal description in GF.

Before diving into the library implementations, let's discuss some peculiar features of Amharic numerals. In Amharic, the ordinal numbers are formed from the cardinal numbers by adding the suffix **-ኛ** / *-ääňňa* after the stem consonant.

Example	Cardinal	Ordinal
	ሁለት	ሁለት+ -ኛ -> ሁለተኛ
	hulät / two	*hulät-ääňňa/ second*

The compound numerals, like English are put separately.

Example ሁለት መቶ ሰላሳ አንድ
hulät mäto sälasa and / two hundred thirty one

ሁለት መቶ ሰላሳ አንደኛ
hulät mäto sälasa and-ääňňa / two hundred thirty first

In addition to species and case, both the ordinals and cardinals inflect in gender and number very much following the pattern of adjectives that modify nouns. This leaves 40 distinct ways of inflecting both cardinal and ordinals when accounting gender (2), number (2), definiteness (2)

and case (4) for each of them. The rest 24 are common cases for plural numbers and singular indefinites. Our system gives this for both the text numerals and digits. Below in the table are shown parts of the the text and digit ordinals for threehundred - ሶስት መቶ -*"sost meto"* - 300.

Masc Sg Def Nom : 3 0 0ኛዉ	Masc Sg Def Nom : ሶስት መቶኛዉ
Masc Sg Def Acc : 3 0 0ኛዉን	Masc Sg Def Acc : ሶስት መቶኛዉን
Masc Sg Def Gen : የ3 0 0ኛዉ	Masc Sg Def Gen : የሶስት መቶኛዉ
Masc Sg Def Dat : ለ3 0 0ኛዉ	Masc Sg Def Dat : ለሶስት መቶኛዉ
…	…

The numerals are basically implemented in the `NumeralsAmh` module. An inflection table of numerals has the following type together with the attributes that a number can inflect in:

```
param
           CardOrd = NCard | NOrd ;
oper
Numeral = {
              s:CardOrd=>Gender=>Number=>Species=>Case=>Str
           };
```

Because of the uniformity of inflectional forms across all ranges of numbers, in Amharic, we do not need to consider special sizes to implement the rules governing their inflections. This uniformity makes syntactic formations using the numerals, such as noun phrases, a lot easier. We do, however, handle the orthographic changes that occur while adding affixes during inflection. One such function can be :

```
oper
regOrd : Str -> Str = \number ->
     case last number of {
                   "ዱ"  => init number + "ደኛ" ;
                   "ት"  => init number + "ተኛ" ;
                   "ኝ"  => init number + "ነኛ" ;
                   "ር"  => init number + "ረኛ" ;
                    _   => number + "ኛ"
      } ;
```

The function helps to handle the formation of ordinal numerals from their cardinal counterparts accounting for the orthographic changes. It describes that whenever the last character of a string `'number'` ends in the shown letters it takes the initials of the `'number'` without the last

character and adds the respective suffixes, otherwise, it just adds "ኛ" /ňa on the original number. If we make use of such a function on strings such as አንድ ,ሁለት , and መቶ, the results are አንደኛ, ሁለተኛand መቶኛ which are equivalents to the English *first*, *second* and *hundredth* respectively.
The abstract syntax in the library details the categories and functions governing the numbers. We show these for numbers less than ten below:

```
cat

   Digit ;           -- 2..9
   Sub10 ;           -- 1..9

fun
   n2, n3, n4, n5, n6, n7, n8, n9 : Digit ;
   pot01 : Sub10 ;                   -- 1
   pot0 : Digit -> Sub10 ;           -- d * 1
```

As can be seen the number 1 is treated separately from the remaining digits . In our concrete module, the following is the type of the categories above in Amharic (the concrete syntax):

```
param

   DForm = unit | ten ;

lincat
  Digit =
{s : DForm => CardOrd => Gender=>Number=>Species=>Case=> Str} ;
  Sub10 =
{s : DForm => CardOrd =>Gender=>Number=>Species=>Case=> Str } ;
```

The inflection table shows what we discussed earlier, that Amharic numbers (both numerals and ordinals) inflect in gender, number, species (definiteness), and case. The `DForm` is given so as to implement a digit with its multiples of ten. The function below forms the inflection table of the digits by taking the unit, its multiple of ten and the ordinal forms:

```
oper

mkNum : Str -> Str -> Str -> {s : DForm =>
         CardOrd=>Gender=>Number=>Species=>Case=> Str} =
    \hulet,haya, huleteNa ->
    {
         s = table {
                   unit => table {
                                   NCard => adjaffix hulet ;
```

```
                                    NOrd =>  adjaffix huleteNa
                                } ;
                    ten  => \\c =>   mkCard c haya
                    }
    } ;
```

The `adjaffix` is the main inflecting function that takes a string to give the 64 possible tabular outputs, while the other operation `mkCard` considers cardinality and a string to make the correct inflections. At this stage we refrain from going into the details of such implementation operation but rather show how such helping functions are finally put to use.

```
lin n2 = mkNum "ሁለት"  "ሃያ" "ሁለተኛ" ;
lin n3 = mkNum "ሶስት"  "ሰላሳ" "ሶስተኛ";
lin n4 = mkNum "አራት"  "አርባ" "አራተኛ";
lin n5 = mkNum "አምስት"  "ሃምሳ" "አምስተኛ";
lin n6 = mkNum "ስድስት"  "ስድሳ" "ስድስተኛ";
lin n7 = mkNum "ሰባት"  "ሰባ" "ሰባተኛ";
lin n8 = mkNum "ስምንት" "ሰማንያ" "ስምንተኛ";
lin n9 = mkNum "ዘጠኝ"  "ዘጠና" "ዘጠነኛ";
lin pot01 = mkNum "አንድ" "አስር" "አንደኛ"  ;
lin pot0 d = d  ;
```

Of course the above does not account for the variations that could arise due to the possibilities of using more than one symbolic character to write the same number. For instance if we are to write 'twenty' in Amharic, ሃያ/ሐያ/ሀያ/ኃያ (all of which are read as 'haya') can be used. This is because the first letters ሃ, ሀ, ሐ and ኃ (all of which are read as 'ha') are given in the fidäl in four different positions with their distinct shapes, even though all sound the same.

The abstract syntax divides the categories based on the decimal system. If we consider the last function in the linearization, `pot0`, is used to transfer the `Digit` into `Sub10` so that it can be used as any numeral less than ten later. The linearization of the rest of the categories from the abstract syntax and their implementation in Amharic follows similar trend as the above. We conclude the section by giving examples of two ordinal forms with similar paramours but gender. We try to clarify this by using an example sentence generated by our system.

Inflection form of the ordinal: `sMasc Sg Def Nom`:ሶስት መቶኛዉ / the three hundredth+masc

*Usage:*preceeds a masculine noun while forming phrases such as noun phrases. A digital form of the same is also possible in our system:`s Masc Sg Def Nom`:300ኛዉ / the 300th+Masc

Example: ሶስት መቶኛው ልጅ በለሶቹን በላ። / the three hundredth boy ate the apples

sost mätoňňaw lǝj bäläsoccun bäla

Inflection form of the ordinal: `s Fem Sg Def Nom` ： ሶስት መቶኛዋ / the three hundredth+fem

Usage: preceeds a feminine noun while forming phrases such as noun phrases. The digital take this form: `s Fem Sg Def Nom` ： 300ኛዋ /the 300th +fem

Example: ሶስት መቶኛዋ ልጃገረድ በለሶቹን በላች ። / The three hundredth girl ate the apples.

sost mätoňňawa lᵊjagäräd bäläsoccun bälacc

5.5 Swadesh[1] Lexicon

The test lexicon we have uses is in `LexiconAmh` module. Below are given examples from the main word types that show how a lexicographer can use this to enter them:

Verbs

Bi-radical

```
break_V2 = mkV3gdl "sbr" ;
sing_V = mkV3mls "zmr";
```

Tri-radical

```
sew_V = mkV2bl "sf" ;
stand_V = mkV2nr "qm";
give_V3 = mkV3 (mkV2sT "sT" ) (mkPrep "ለ" "" True);
```

Quadri-radical

```
freeze_V = mkV4dbdb "qzqz";
throw_V2 = mkV4dbdb "wrwr";
```

Nouns

```
ship_N = mkN[2] "መርከብ" feminine ;
church_N = mkN (mkN "ቤተ") (mkN "ክርስትያን" feminine);
ceiling_N = mkN "ጣራ";
```

Adjectives

```
full_A = mkA "ሙሉ" ;
heavy_A = mkA "ከባድ" ;
```

We also provide additional list of structural words in the `StructuralAmh` module for instance,

```
we_Pron = pronNP "እኛ" "እኛን" "የእኛ" "ለእኛ" (Per1 Pl);
```

[1] The Swadesh list is one of several lists of vocabulary with basic meanings, developed by Morris Swadesh.

[2]This single wrapper function mkN is overloaded with the various forms of nominal declension to make it 'smart' enough to guess the outputs for the different input signatures. Such functions in GF are called smart paradigms.

where the helping function pronNP is defined in ResAmh to take four strings and person-number-gender agreement feature, to build a noun phrase with the proper cases,i.e., nominative, accusative ,genitive and dative. This can be shown below.

```
pronNP : (N,A,G,D : Str) -> PerNumGen -> NP = \N,A,G,D,png-> {
          s = table {
                          Nom => N ;
                          Acc => A ;
                          Gen => G ;
                          Dat => D

                          } ;

                        a = Agr
                      };
```

Similarly, the rest of the structural words which includes, prepositions, conjunctions, adverbs, quantifiers and so on are constructed by making use of the proper functions we provide in the `ResAmh` module.

5.6 Syntax

The final part of the grammar that we have implemented is a set of syntactic rules such as the one we described in section in chapter 3.2. A complete description of the language independent API rules in the abstract resource grammar,i.e., the syntactic structures of the abstract API is given by Ranta in his description of the resource library (Ranta, 2004), so we do not discuss them here. We give two examples, one noun determination and another verb predication, to show the expressive power of the formalism. We as well forward some points with regard to the VPSlash (a verb phrase without an object / complement) and verb complementation issues, at last the uses of conjunction in order to coordinate the semantic flow of two or more list of expressions in the same category is given.

Example 1:

The example we give hereunder is the formation of Amharic determiner using quantifiers and numerals to show the concepts of definite and indefinite articles and they are implemented. First let's give the abstract categories (`cat`) and functions (`fun`). Our goal is to linearize the rules, both categories (using `lincat`) and functions (using `lin`) into Amharic according to the grammar.

```
cat
          Det;
          Quant;
          Num;
fun
```

```
          DetQuant   : Quant -> Num -> Det ;
          IndefArt   : Quant ;
          DefArt     : Quant ;
          NumSg      : Num ;
          NumPl      : Num ;

lincat
           Quant = {
                   s : Number  => Gender => Case => Str;
                   d : Species;
               isNum : Bool;
               isPron: Bool
                     } ;
          Det  = {
                   s : Gender => Case => Str ;
                   d : Species;
                   n : Number;
               isNum : Bool;
              isPron : Bool
                     } ;

          Num  = { s : Species=>Case => Str ;
                   n : Number ;
             hasCard : Bool
                     } ;

lin
          DetQuant quant num = {
          s = \\g,c => quant.s!num.n!g!c ++ num.s!quant.d!c ;
          d = quant.d;
          n = num.n;
          isNum = True;
          isPron = quant.isPron
          } ;

          DefArt = {
          s = \\_,_,_ => [];
          d = Def ;
          isNum,isPron = False
          } ;

          IndefArt = {
          s = \\n,g,_ =>

          case <n,g> of {
                   <Sg,Masc> => "አንድ" ++ [];
                   〈Sg,Fem〉   => "አንዲት" ++ [];
                   <Pl,_>    => [] };
```

```
d = Indef ;
isNum,isPron = False
} ;
```

One way of making determiners such as *these five,* is by making use of quantifiers and numerals as given by `DetQuant`. It can be seen from the same linearization that the inherent number of the determiner comes from the numeral while the definiteness is decided by the quantifier. Our interest is to show the extra things that come with Amharic definite and indefinite articles and how they could be captured by the `Quant` category. From `DefArt` it can be seen that the definite article in Amharic has no explicit word corresponding to it making its implementation look more complex. But there is another way out of this, i.e., treating the quantifier's inherent species as definite regardless of the number, gender and case. This better explains the situation with Amharic definiteness – a point of discussion amongst scholars (Beermann and Ephrem, 2007). The other feature could be the case of the indefinite article for singular quantifiers. Indefinite nouns are normally not marked with determiners and hence we could leave the nucleus quantifier the same as definite article while only marking the species with `Indef`. But occasionally the cardinal numeral *and*/ አንድ (‘one’) is used to indicate the meaning of indefiniteness and that is what we have shown in `IndefArt`. It is further to be noted that the indefinite article takes two different forms depending on the gender. For instance, in Amharic, we say አንድ ልጅ *and lɔj* / a boy and አንዲት ልጃገረድ *andit lɔjagäräd* to mean *‘a girl’*.

To summarize this, we show two sentences that make use of verb phrase from the verb 'love' (we shall give the verb predication implementation in our next example) and a noun phrase which in turn is made by such determiners as the one shown in the previous example and nouns (boy and girl).

ልጁ አንዲት ልጃገረድን ይወዳል።
lɔju andit lɔjagäräd∂n y∂wädal
The boy a girl loves : the boy loves a girl

ልጃገረዷ አንድ ልጅን ትወድዳለች።
lɔjagärädua and lɔj∂n t∂wädaläcc
The girl a boy loves : the girl loves a boy

These two simple sentences could show how much Amharic deviates from English. English only has to swap the nouns 'girl' and 'boy' to bring the required semantic difference while there is high similarity between the structures for the two cases. Meanwhile, in Amharic we have to consider lots of grammatical decisions almost on every word of the sentence. The way we explain definiteness, the way we give indefiniteness, the way we decline for accusative cases and even the way we inflect the verbs are all different between the two sentences.

Example 2:

As a second example let's take the rule of verb predication where a verb phrase is formed from a verb. In order to do so we need to decide the type of verb phrases (VP) first. A verb phrase in Amharic can be given in its compact form as:

```
param

          TenseAmh=PresFut|SimplePast|PresPerf|PastPerf|PresCont|
                   PastCont;

          Polarity = Pos | Neg;
lincat

          VP = {
                s : TenseAmh=> Polarity =>PerNumGen => Str ;
            compl : NP
                  } ;
```

In our implementation the `VP` takes several more inherent features, which we have summarized and put just as an `NP` complement for the sake of brevity. It can also be seen from the table that a `VP` inflects for tense, polarity (which is used when making either positive or negative sentences) and agreement feature `PerNumGen`

The operation `predV` below takes a verb and makes a verb phrase. Most of the discussions we had in the verb morphology are now implemented here (see that we have used the transliteration we described earlier whose list can also be found in the appendix A):

```
predV :V -> VP = \v ->
          {

          s = \\t,p,png =>

          let

          ketebku             =    v.s ! Perf !Act! png ;
          eketbalehu          =    v.s ! Imperf !Act! png ;
          keteb               =    v.s ! Jus_Imperat !Act! png ;
          ketbie              =    v.s ! Gerund !Act! png ;
          mekteb              =    v.s ! Infinitive !Act! png ;
          ketabi              =    v.s ! Parti !Act! png ;
          ketbiealehu         =    v.s ! CompPerf !Act! png ;
          eketib              =    v.s ! Cont !Act! png ;

          in
            case <t,p> of
          {
          <PresFut,Pos>  => eketbalehu ;
```

```
          <PresFut,Neg>    => "!" ++"&+"++ eketib ++"&+" ++"m" ;

          <PresPerf,Pos>   => ketbiealehu;

          <PresPerf,Neg>   => "!l"++"&+"++ketebku++"&+"++"m";

          <PresCont,Pos>   => "(y'"++ "&+"++ketebku ++"n'w" ;

          <PresCont,Neg>   =>"(y'"++ "&+"++ketebku ++"Ayd'l'm" ;

          <SimplePast,Pos> => ketebku;

          <SimplePast,Neg> => "!l"++"&+"++ketebku++"&+"++"m";

          <PastPerf,Pos>   => ketbie++"n'b'r";

<PastPerf,Neg>=>"!l"++"&+"++ketebku++"&+"++"m"++"n'b'r";

          <PastCont,Pos>   => "(y'"++ "&+"++ketebku ++ "n'b'r" ;

          <PastCont,Neg>   => "(y'"++ "&+"++ketebku ++
"Aln'b'r'm"
                        };
          comp      = { s = \\_ => []  };
                    };
```

We want to mark at this point that verb phrases are primarily formed from a verb and its required complements. But verbs can be of various categories corresponding to the possible complements and their combinations. Such division of verbs by the complement is known as subcategorization. (Ranta, 2009). The way such subcategorizations are handled is abstract in a sense that it doesn't distinguish prepositional phrase complements from noun phrases. This means that the treatment given to the prepositional phrases is almost the same as noun phrases. The case is a language-dependent concrete syntax feature, and complement cases are inherent features of verbs. For instance, the linearization type of V3 is formed from the linearization type of one-place verbs by adding two cases. In Amharic, these cases are simply prepositions, which can be expressed as strings. Thus the verb 'talk' (to somebody about something) has the following linearization

```
mkV3 (mkV3asr "'wr")(mkPrep "ከ" "ጋር" True)(mkPrep "ስለ" "" True);
```

At this point, we do not intend to explain the code and how it evolved to get to this form in this report but merely mention the fact that the prepositional phrases are almost treated the same way as noun phrase complements in GF.

Clauses and Sentences

A clause in GF, which is a syntactic category that has variable tense, polarity and order, can be formed in various ways. Of such formations, predication of a NP and VP is one as shown below.

```
fun PredVP    : NP -> VP -> Cl
```

Where the clause is given by

```
lincat Cl : Type = { s : TenseAmh => Polarity => Str};
```

The GF tense system found in the common API, which is given as tense and anteriority, is a simplified one. There are only four tenses named as present, past, future and conditional, and two possibilities of anteriority (Simul, Anter) which leave the possibility of generating only 8 possible combinations. This coverage is not sufficient enough to handle all the tense forms of Amharic which we mentioned earlier, therefore, further coverage is given at the clause level to address that.

Further explaining this problem, the above `PredVP` function creates clauses which can further be fixed for tense and polarity to form sentences. One such sentence forming function that can fix clauses for polarity and tense is:

```
fun UseCl    : Temp -> Pol -> Cl  -> S
```

While forming declarative sentences, the tense in the Temp category refers to abstract level tense and we just map it to Amharic tenses by selecting the appropriate clause as shown below.

```
UseCl  t ap cl =
               let ss : Str = case t.t of
                        {
                         Pres => cl.s ! PresFut ! ap.p ;
                         Cond => cl.s ! PresFut ! ap.p ;
                         Past => cl.s ! SimplePast ! ap.p ;
                         Fut  => cl.s ! PresFut ! ap.p
                        }
                     in {

                   s =  ss
              };
```

Here we may lose some semantic equivalence as, for example, there is no conditional tense in Amharic and what we picked instead is the one that is 'similar' to this tense type, `PresFut`[1] which may not be correct all the time.

Question Forms

Unlike some of the languages in the GF library, Amharic clauses do not vary in 'order' (direct or question). We consider only direct clauses even when forming interrogative sentences. The way the speaker says the sentence and the addition of tags like "እንዴ"– 'ände' at the end of the direct clause that was formed from the clause level helps to make questions. For examples a simple way to create a question clause can be given in the abstract as:

```
fun QuestCl      : Cl -> QCl ;
```

which can be linearized as

```
QuestCl cl = {
            s = \\t,p => cl.s! t! p ++ "እንዴ"
          };
```

The tense and polarity of this clause will be fixed at the sentence level while forming question sentences the same way we did with the normal clauses that use UseCl as described above.
There are also other forms of question clauses which include clauses made with interrogative pronouns (IP), interrogative adverbs (IAdv), and interrogative determiners (IDet) which are given as structural words.

Coordinating Conjunction

A coordinating conjunction can join lists of expression that a user of a language wants to emphasize equally. The list can be composed of 2 (X and Y) or more (X, Y, Z and U). The GF library allows coordination of five categories (Ranta, 2009)

- አ adverbs (here or there),
- አ adjectival phrases (cold and warm),
- አ noun phrases (she or John),
- አ relative clauses (who walks or whom she loves), and
- አ sentences (he walks and she runs)

The conjunctions are given as a structural word in the structuralAmh.gf module. These can assume wither a simple (and, or) or discontinuous (both-and, either-or) forms. We show below the implementation of coordination in the NP category eg. "she and we" .

[1] *The present and future tenses are not separated in Amharic and we give them as PresFut.*

```
fun       ConjNP : Conj ->[NP]->NP;
```

Such a rule can be applied to lists of two or more elements as shown in the translation given by our GF system.

Languages: LangAmh LangEng

Lang> p -lang=LangEng **"he , John and she eat red apple"**| l -lang=LangAmh -treebank -to_amharic
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP (ConjNP and_Conj (ConsNP (UsePron he_Pron) (BaseNP (UsePN john_PN) (UsePron she_Pron)))) **(ComplSlash (SlashV2a eat_V2) (MassNP (AdjCN (PositA red_A) (UseN apple_N))))**))) NoVoc
LangAmh: **እርሱ ፣ ዮሃንስ እና እርሷ ቀይ በለስ ይበላሉ**

The "፣" in the Amharic translation by GF is an equivalent mark for "," in English. It can also be seen that the verb selection in the sentence smartly takes its agreement feature as third person plural, i.e., as in 'they eat red apple'.

A slash category CSlash is like the category C but "missing" a noun phrase.(Ranta 2009). From the bold written part of the parse tree structure provided by GF, the function SlashV2a is given in the common API as:

```
fun: SlashV2a : V2 ->VPSlash
```

It takes a two place verb `V2` (**ይበላሉ** *-yəbälalu* eat-*Per3Pl*) to form a verb phrase known as `VPSlash` which lacks a complement NP. At a later stage this VPSlash is converted to VP through `ComplSlash` function which is given in abstract syntax as:

```
fun: ComplSlash : VPSlash -> NP -> VP ;
```

where as in this case the `NP` (**ቀይ በለስ** - *qäy bäläs* - red apple) is formed by `MassNP` that simple takes a noun `(CN)` piped from `AdjCN` which takes `Adj` **ቀይ**- *qäy-red* and another noun `CN` **በለስ** - *bäläs*- apple as inputs as shown below.

```
fun MassNP : CN ->NP
fun AdjCN   : AP -> CN  -> CN
```

Furthermore, in Amharic the complement of a verb precedes the actual verb e.g (**እርሷ መሄድ ትፈልጋለች** *ə*rsua mähed t*ə*fäl*ə*galäc "she wants to go"), where (**ትፈልጋለች** - t*ə*fäl*ə*galäc "want"-*Per3SgFem*) is a complement of verb (**መሄድ**-mähed "go"-*inf*).

Chapter 6

Related Work

Amharic is an under resourced language with regard to NLP. Therefore, works regarding the computational aspect of the language are rather hard to come by. This being the general case, there are initiatives by individual researchers whose theoretical works extend from machine translation to speech recognition. Even if such islands of researches are still shy of coordination and practicality to bring the language a step closer to the world of computation, we found the following works worth mentioning as a related work.

6.1 Arabic Resource Grammar in Grammatical Framework

Our work on Amharic has benefited much from the work of Ali El Dada and Aarne Ranta on the implementation of the other Semitic language in the library, Arabic. (El Dada and Ranta, 2007) Their work deals with some of the problems in the NLP of Arabic and introduces natural language constructs and rules implemented in libraries. This has considered rules that span from orthography and morphology to syntax. Similarly, we revised another mini resource work on Hebrew and Maltese (Dannélls and Camiller, 2010).

6.2 *HornMorpho* Amharic Morphological Analyzer and Generator

HornMorpho[1] is a program that analyzes Amharic words into their constituent morphemes (meaningful parts), returning, whenever possible, the stem or root of the word, along with a representation of its grammatical structure. It is the first in a planned series of programs to handle the morphology of Ethiopian and Eritrean languages.

The software is released as part of the L3 project ("Learning Lots of Languages") which has a long-term goal of developing a system to translate to and from many under-represented languages and (less ambitiously) of creating tools to be used in information retrieval and computer-assisted language learning with these languages.

The approach they implement is based on the Finite State Transducers weighted with feature structures. The program is written in Python and the FST module they use is adapted from the Natural Language Tool Kit (NLTK)[2] - an open-source suit of Python tools for computational linguistics applications. The generator takes inputs both in Romans and Fidäl (UTF-8 encoding)

[1] https://www.cs.indiana.edu/~gasser/software.html
[2] www.nltk.org

versions and the orthographic version of the analyzer romanizes the Fidäl inputs using the SERA transliteration convention.
Even if the morphology part of our study is limited to inflectional generations, there is a crossing with FST approach they implemented in a way that we both adopt the root-pattern, where the roots and patterns are explicitly separated for combining during processing. However, the FST implementation of HornMorpho adds features such as weights on the transitions and uses multi layered transducers to do the analysis and generation.(Gasser, 2009)

6.3 Syntax

Works on Amharic syntax, even at theoretical level, are rather rare to find than those on morphology. Of the available works, we note Baye Yimam's work (Yimam, 1987) that gives a formal description of the Amharic grammar in his book written in Amharic. His paper on the interaction of tense, aspect, and agreement in Amharic syntax is another piece to mention (Yimam, 2006). A thesis work by Atalech, presently a Phd student at Stockholm University, (Argaw, 2002) describes a work on automatic sentence parsing for Amharic text. Finally, works on dependency parsers for Amharic by Gasser (Gasser, 2010) where he introduces a grammar for a fragment of Amharic within the Extensible Dependency Grammar (XDG) framework gives a highlight into some of the morphological and syntactic complexities in the language and this,i.e XDG, methodology to deal with them.

Chapter 7

Conclusion

We discussed in this report the implementation of Amharic as a resource grammar. The project consists of 25 modules and more than 4,000 lines of GF code that cover orthographic, morphological and syntactical features of the language.

We took on the project from the zero-ground all the way up. Our work started with the morphology where majority of the nominal declensions followed uniform and simple trends. The verbal inflections, though not complete, required us to design unique patterns longer than 1,000 lines of code. We approached such verbal analysis using transliterations, unlike nouns and adjectives that use Amharic characters, as it gave us flexibility and lessened the burden of orthographic errors dramatically. The numerals from a previous work had to get refurnished and can now be used smoothly. At times we used near-meanings when we miss direct translations and when the rule of the syntax is far too complex to implement. The compiling time is also reasonably good to generate PGF files - portable grammar formats, which will be imported and processed by main stream programming languages like Haskell and Java. (Angelov and Bringert, 2009)

Throughout the project we came to witness that GF indeed has lots of strengths when describing rather complicated linguistic issues such as grammar rules and inflection tables. We note also the extent of classiness in which formalism like GF enables us to represent linguistic rules and abstractions.

Finally, the current grammar do not cover all aspect of Amharic and it does not be used to parse arbitrary texts as we do in Google-Translator®; but with extended inclusion of more syntactic rules and a multi-dimensional overhaul of morphology, we strongly assert that GF has all that it takes to make Amharic be used in text generation applications and for software localization works.

Chapter 8

Future Work

A language with morphosyntatic complexity like Amharic puts immense challenges during its implementation as a resource grammar. We do not dare to claim that our work has covered everything there is to know about Amharic at this stage. We rather aim to extend such an initiative through open sharing of sources to linguists (computational and theoretical) and anyone interested on Amharic NLP. Therefore, we site the following additions and completions as future works:

- The level of coverage given to root-pattern classification of the verbs could further be augmented to address various other derivational templates than we could cover here. Especially verbs that start with a vowel and 'tä' behave in a different way. Therefore they need further analysis than we could make in this work. The implementation of verb 'to have' is also over simplified and needs detailed consideration like the verb to-be's.

- Improving the efficiency and broadening the coverage of the grammar can be another direction for future work. Some forms are also left out for the sake of time. These include the case of object suffixes, relative clauses and aspects such as reciprocals and iteratives for the various TAM's and voices. These will require redesigning some types and patterns. [1]

- Steps towards a robust grammar cannot be attained without the inclusion of a larger lexicon. Therefore, future work should consider more lexica to test the various morphosyntactic properties of the language.

[1] There are also some feature of Amharic which are not yet given a coverage with the current API , such as extra tense varieties and some pronoun forms, these features are implemented in the `ExtraAmh` module.

- Writing an application grammar of some kind (could be started from those that have already used GF such as the phrase book in the MOLTO project[1]) in Amharic can also help test the practicalities of our approach.

- Finally, generation of sentences attained by our system so far should be extended into texts. Such achievements, when implemented on a good amount of lexica, may lead to the generation of a good amount of bi or multi-lingual corpora which are missing in Amharic. These, eventually, could serve as an important input to statistical technologies, such as machine translation, bringing the language into the lights of computation faster.

[1] http://www.molto-project.eu/demo/phrasebook

References

Angelov, Krasimir, Bringert, Björn. and Ranta, Aarne. *PGF: A Portable Run-Time Format for Type-Theoretical Grammars*. Journal of Logic, Language and Information. 2009

Angelov, Krasimir. *Type-Theoretical Bulgarian Grammar.* Proceedings of the 6th international conference on Advances in Natural Language Processing, 2008

Argaw, Atalech. *Automatic Sentence Parsing for Amharic Text: An Experiment Using Probabilistic Context Free Grammars.* Addis Ababa: Addis Ababa University Press. 2002

Bayou, Abiyot. *Design and Development if a Word Parser for Amharic Language: Master's Thesis*. Addis Ababa: Addis Ababa University Press. 2000

Beermann, Dorothee and Ephrem, Binyam: *The Definite Article and Possessive Marking in Amharic*. Texas Linguistics Society 9: Morphosyntax of Underrepresented Languages. CSLI Publications. 2007

Dana Dannélls and John J. Camiller. *Verb Morphology of Hebrew and Maltese: Towards an Open Source Type Theoretical Resource Grammar in GF.* Proceedings of LREC. 2010

El Dada, Ali and Ranta, Aarne. *Implementing an Open Source Arabic Resource Grammar in GF. In Perspectives on Arabic Linguistics XX*. John Benjamins Publishing Company. 2007

Enache, Ramona; Ranta, Aarne and Angelov, Krasimir: *An Open-Source Computational Grammar for Romanian.* In Computational Linguistics and Intelligent Text Processing 11th International Conference, Iasi, Romania 2010

Fisseha, Sissay and Haller, Johann. *Amharic verb lexicon in the context of machine translation* .Saarbrücken: University of Saarland Press. 2003

Gasser, Michael. *A dependency grammar for Amharic*. Workshop on Language Resources and Human Language Technologies for Semitic Languages. 2010

Gasser, Michael. *Semitic morphological analysis and generation using finite state transducers with feature structures.* In Proceedings of the 12th Conference of the European Chapter of the ACL, pages 309–317, Athens. 2009

Grishman, Ralph. *Computational Linguistics: An Introduction*. New York: Cambridge University Press. 1994

Khegai, Janna. *Language Engineering in Grammatical Framework (GF): Phd Thesis.* Göteborg: Chalmers University Press. 2006

Leslau, Wolf. *Reference Grammar of Amharic*. Wiesbaden: Otto Harrassowitz. 1995

Leslau, Wolf. *An Amharic Reference Grammar*. University of California: Los Angelese 1969

Ranta, Aarne. *Grammatical Framework : A Type Theoretical Grammar Formalism*. The Journal of Functional Programming 14(2), 145-189 . 2004

Ranta, Aarne. *The GF Resource Grammar Library.* Linguistic Issues in Language Technology – LiLT . 2009

Ranta, Aarne. *GF: A Multilingual Grammar Formalism*. Language and Linguistics Compass. 2009b

Ranta, Aarne. *Grammatical Framework: A Programming Language for Multilingual Grammars and Their Applications*. Stanford: CSLI Publications, 2010 (to appear)

Yimam, Baye. *The interaction of tense, aspect, and agreement in Amharic syntax . In Selected Proceedings of the 35th Annual Conference on African Linguistics*, pages 193-202, Somerville. 2006.

Yiman, Baye. *የአማርኛ ሰዋሰው.* Addis Ababa: E.M.P.D.A Press. 1987

Appendices

Appendix - A- Transliteration Table for the Fidäl as used in the project.
Format: | *UTF code no* | *fidäl character* | *transliteration in the system* |

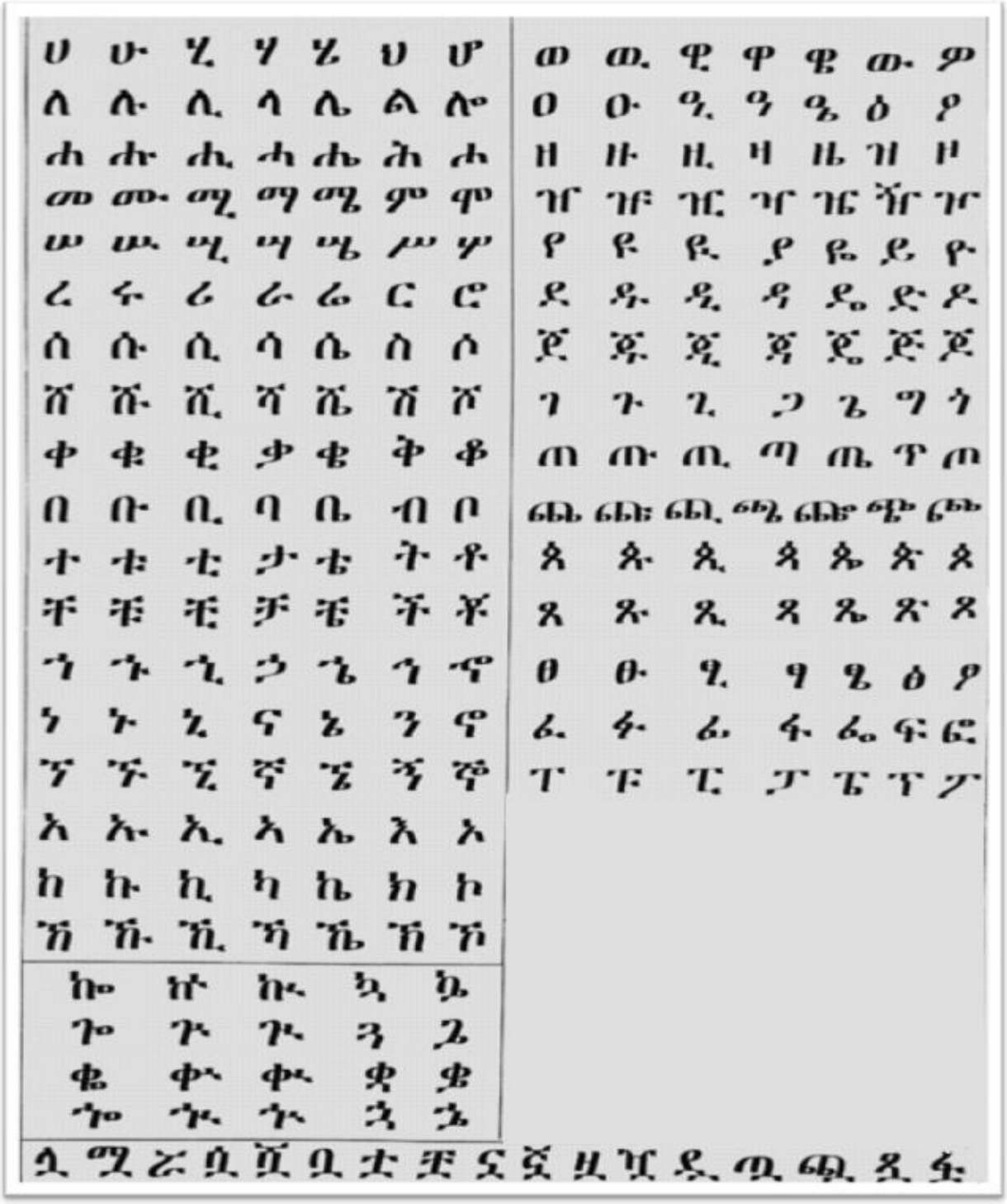
1200	ሀ	h'
1201	ሁ	h&
1202	ሂ	h#
1203	ሃ	h!
1204	ሄ	h%
1205	ህ	h
1206	ሆ	h/
1207	ሇ	h7
1208	ለ	l'
1209	ሉ	l&
120a	ሊ	l#
120b	ላ	l!
120c	ሌ	l%
120d	ል	l
120e	ሎ	l/
120f	ሏ	l7
1210	ሐ	H'
1211	ሑ	H&
1212	ሒ	H#
1213	ሓ	H!
1214	ሔ	H%
1215	ሕ	H
1216	ሖ	H/
1217	ሗ	H7
1218	መ	m'
1219	ሙ	m&
121a	ሚ	m#
121b	ማ	m!
121c	ሜ	m%
121d	ም	m
121e	ሞ	m/
121f	ሟ	m7
1220	ሠ	s'
1221	ሡ	s&
1222	ሢ	s#
1223	ሣ	s!
1224	ሤ	s%
1225	ሥ	s
1226	ሦ	s/
1227	ሧ	s7
1228	ረ	r'
1229	ሩ	r&
122a	ሪ	r#
122b	ራ	r!
122c	ሬ	r%
122d	ር	r
122e	ሮ	r/
122f	ሯ	r7
1238	ሸ	x'
1239	ሹ	x&
123a	ሺ	x#
123b	ሻ	x!
123c	ሼ	x%
123d	ሽ	x
123e	ሾ	x/
123f	ሿ	x7
1240	ቀ	q'
1241	ቁ	q&
1242	ቂ	q#
1243	ቃ	q!
1244	ቄ	q%
1245	ቅ	q
1246	ቆ	q/
1247	ቇ	q7
1260	በ	b'
1261	ቡ	b&
1262	ቢ	b#
1263	ባ	b!
1264	ቤ	b%
1265	ብ	b
1266	ቦ	b/
1267	ቧ	b7
1268	ቨ	v'
1269	ቩ	v&
126a	ቪ	v#
126b	ቫ	v!
126c	ቬ	v%
126d	ቭ	v
126e	ቮ	v/
126f	ቯ	v7
1270	ተ	t'
1271	ቱ	t&
1272	ቲ	t#
1273	ታ	t!
1274	ቴ	t%
1275	ት	t
1276	ቶ	t/
1277	ቷ	t7
1278	ቸ	c'
1279	ቹ	c&
127a	ቺ	c#
127b	ቻ	c!
127c	ቼ	c%
127d	ች	c
127e	ቾ	c/
127f	ቿ	c7
1280	ኀ	X'
1281	ኁ	X&
1282	ኂ	X#
1283	ኃ	X!
1284	ኄ	X%
1285	ኅ	X
1286	ኆ	X/
128b	ኋ	X7
1290	ነ	n'
1291	ኑ	n&
1292	ኒ	n#
1293	ና	n!
1294	ኔ	n%
1295	ን	n
1296	ኖ	n/
1297	ኗ	n7
1298	ኘ	N'
1299	ኙ	N&
129a	ኚ	N#
129b	ኛ	N!
129c	ኜ	N%
129d	ኝ	N
129e	ኞ	N/
129f	ኟ	N7
12a0	አ	!
12a1	ኡ	&
12a2	ኢ	#
12a3	ኣ	A
12a4	ኤ	%
12a5	እ	(
12a6	ኦ	/
12a7	ኧ	'
12a8	ከ	k'
12a9	ኩ	k&
12aa	ኪ	k#
12ab	ካ	k!
12ac	ኬ	k%
12ad	ክ	k
12ae	ኮ	k/
12b3	ኳ	k7
12c8	ወ	w'
12c9	ዉ	w&
12ca	ዊ	w#
12cb	ዋ	w!
12cc	ዌ	w%
12cd	ው	w
12ce	ዎ	w/
12cf	ዏ	w7
12d0	ዐ	ä'
12d1	ዑ	ä&
12d2	ዒ	ä#
12d3	ዓ	ä!
12d4	ዔ	ä%
12d5	ዕ	ä
12d6	ዖ	ä/
12d8	ዘ	z'
12d9	ዙ	z&
12da	ዚ	z#
12db	ዛ	z!
12dc	ዜ	z%
12dd	ዝ	z
12de	ዞ	z/
12df	ዟ	z7
12e0	ዠ	Z'
12e1	ዡ	Z&
12e2	ዢ	Z#
12e3	ዣ	Z!
12e4	ዤ	Z%
12e5	ዥ	Z
12e6	ዦ	Z/
12e7	ዧ	Z7
12e8	የ	y'
12e9	ዩ	y&
12ea	ዪ	y#
12eb	ያ	y!
12ec	ዬ	y%
12ed	ይ	y
12ee	ዮ	y/
12ef	ዯ	y7
12f0	ደ	d'
12f1	ዱ	d&
12f2	ዲ	d#
12f3	ዳ	d!
12f4	ዴ	d%
12f5	ድ	d
12f6	ዶ	d/
12f7	ዷ	d7
1300	ጀ	j'
1301	ጁ	j&
1302	ጂ	j#
1303	ጃ	j!
1304	ጄ	j%
1305	ጅ	j
1306	ጆ	j/
1307	ጇ	j7
1308	ገ	g'
1309	ጉ	g&
130a	ጊ	g#
130b	ጋ	g!
130c	ጌ	g%
130d	ግ	g
130e	ጎ	g/
1313	ጓ	g7
1320	ጠ	T'
1321	ጡ	T&
1322	ጢ	T#
1323	ጣ	T!
1324	ጤ	T%
1325	ጥ	T
1326	ጦ	T/
1327	ጧ	T7
1328	ጨ	C'
1329	ጩ	C&
132a	ጪ	C#
132b	ጫ	C!
132c	ጬ	C%
132d	ጭ	C
132e	ጮ	C/
132f	ጯ	C7
1330	ጰ	P'
1331	ጱ	P&
1332	ጲ	P#
1333	ጳ	P!
1334	ጴ	P%
1335	ጵ	P
1336	ጶ	P/
1337	ጷ	P7
1338	ጸ	S'
1339	ጹ	S&
133a	ጺ	S#
133b	ጻ	S!
133c	ጼ	S%
133d	ጽ	S
133e	ጾ	S/
133f	ጿ	S7
1348	ፈ	f'
1349	ፉ	f&
134a	ፊ	f#
134b	ፋ	f!
134c	ፌ	f%
134d	ፍ	f
134e	ፎ	f/
134f	ፏ	f7
1350	ፐ	p'
1351	ፑ	p&
1352	ፒ	p#
1353	ፓ	p!
1354	ፔ	p%
1355	ፕ	p
1356	ፖ	p/
1357	ፗ	p7

Appendix – B1 - Simple inflection table of the verb *sbr (break)*

Format : *Verb form Voice (Person Number Gender) : Verb in Amharic Eg: Perf Act (Per1 Sg): ሠበርኩ - I broke .*

Perf Act (Per1 Sg) : ሠበርኩ
Perf Act (Per1 Pl) : ሠበርን
Perf Act (Per2 Sg Masc) : ሠበርክ
Perf Act (Per2 Sg Fem) : ሠበርሽ
Perf Act (Per2 Pl Masc) : ሠበራችሁ
Perf Act (Per2 Pl Fem) : ሠበራችሁ
Perf Act (Per3 Sg Masc) : ሠበረ
Perf Act (Per3 Sg Fem) : ሠበረች
Perf Act (Per3 Pl Masc) : ሠበሩ
Perf Act (Per3 Pl Fem) : ሠበሩ
Perf Pas(Per1 Sg) : ተሠበርኩ
Perf Pas(Per1 Pl) : ተሠበርን
Perf Pas(Per2 Sg Masc) : ተሠበርክ
Perf Pas(Per2 Sg Fem) : ተሠበርሽ
Perf Pas(Per2 Pl Masc) : ተሠበራችሁ
Perf Pas(Per2 Pl Fem) : ተሠበራችሁ
Perf Pas(Per3 Sg Masc) : ተሠበረ
Perf Pas(Per3 Sg Fem) : ተሠበረች
Perf Pas(Per3 Pl Masc) : ተሠበሩ
Perf Pas(Per3 Pl Fem) : ተሠበሩ
Imperf Act (Per1 Sg) : ዕሠብራለሁ
Imperf Act (Per1 Pl) : ዕንሠብራለን
Imperf Act (Per2 Sg Masc) : ትሠብራለህ
Imperf Act (Per2 Sg Fem) : ትሠብርያለሽ
Imperf Act (Per2 Pl Masc) : ትሠብራላችሁ
Imperf Act (Per2 Pl Fem) : ትሠብራላችሁ
Imperf Act (Per3 Sg Masc) : ይሠብራል
Imperf Act (Per3 Sg Fem) : ትሠብራለች
Imperf Act (Per3 Pl Masc) : ይሠብራሉ
Imperf Act (Per3 Pl Fem) : ይሠብራሉ
Imperf Pas(Per1 Sg) : ዕሠበራለሁ
Imperf Pas(Per1 Pl) : ዕንሠበራለን
Imperf Pas(Per2 Sg Masc) : ትሠበራለህ
Imperf Pas(Per2 Sg Fem) : ትሠበርያለሽ
Imperf Pas(Per2 Pl Masc) : ትሠበራላችሁ
Imperf Pas(Per2 Pl Fem) : ትሠበራላችሁ
Imperf Pas(Per3 Sg Masc) : ይሠበራል
Imperf Pas(Per3 Sg Fem) : ትሠበራለች
Imperf Pas(Per3 Pl Masc) : ይሠበራሉ
Imperf Pas(Per3 Pl Fem) : ይሠበራሉ
Jus_Imperat Act (Per1 Sg) : ልሥበር
Jus_Imperat Act (Per1 Pl) : ዕንሥበር
Jus_Imperat Act (Per2 Sg Masc) : ሥበር
Jus_Imperat Act (Per2 Sg Fem) : ሥበሪ
Jus_Imperat Act (Per2 Pl Masc) : ሥበሩ
Jus_Imperat Act (Per2 Pl Fem) : ሥበሩ
Jus_Imperat Act (Per3 Sg Masc) : ይሥበር
Jus_Imperat Act (Per3 Sg Fem) : ትሥበር
Jus_Imperat Act (Per3 Pl Masc) : ይሥበሩ
Jus_Imperat Act (Per3 Pl Fem) : ይሥበሩ
Jus_Imperat Pas(Per1 Sg) : ልሠበር
Jus_Imperat Pas(Per1 Pl) : ዕንሠበር
Jus_Imperat Pas(Per2 Sg Masc) : ተሠበር
Jus_Imperat Pas(Per2 Sg Fem) : ተሠበሪ
Jus_Imperat Pas(Per2 Pl Masc) : ተሠበሩ
Jus_Imperat Pas(Per2 Pl Fem) : ተሠበሩ
Jus_Imperat Pas(Per3 Sg Masc) : ይሠበር
Jus_Imperat Pas(Per3 Sg Fem) : ትሠበር
Jus_Imperat Pas(Per3 Pl Masc) : ይሠበሩ
Jus_Imperat Pas(Per3 Pl Fem) : ይሠበሩ
Gerund Act (Per1 Sg) : ሠብሬ
Gerund Act (Per1 Pl) : ሠብረን
Gerund Act (Per2 Sg Masc) : ሠብረክ
Gerund Act (Per2 Sg Fem) : ሠብረሽ
Gerund Act (Per2 Pl Masc) : ሠብራችሁ
Gerund Act (Per2 Pl Fem) : ሠብራችሁ
Gerund Act (Per3 Sg Masc) : ሠብሮ
Gerund Act (Per3 Sg Fem) : ሠብራ
Gerund Act (Per3 Pl Masc) : ሠብረው
Gerund Act (Per3 Pl Fem) : ሠብረው
Gerund Pas(Per1 Sg) : ተሠብሬ
Gerund Pas(Per1 Pl) : ተሠብረን
Gerund Pas(Per2 Sg Masc) : ተሠብረክ
Gerund Pas(Per2 Sg Fem) : ተሠብረሽ
Gerund Pas(Per2 Pl Masc) : ተሠብራችሁ
Gerund Pas(Per2 Pl Fem) : ተሠብራችሁ
Gerund Pas(Per3 Sg Masc) : ተሠብሮ
Gerund Pas(Per3 Sg Fem) : ተሠብራ
Gerund Pas(Per3 Pl Masc) : ተሠብረው
Gerund Pas(Per3 Pl Fem) : ተሠብረው
Infinitive Act (Per1 Sg) : መሥበር
Infinitive Act (Per1 Pl) : መሥበር
Infinitive Act (Per2 Sg Masc) : መሥበር
Infinitive Act (Per2 Sg Fem) : መሥበር
Infinitive Act (Per2 Pl Masc) : መሥበር
Infinitive Act (Per2 Pl Fem) : መሥበር
Infinitive Act (Per3 Sg Masc) : መሥበር
Infinitive Act (Per3 Sg Fem) : መሥበር
Infinitive Act (Per3 Pl Masc) : መሥበር
Infinitive Act (Per3 Pl Fem) : መሥበር
Infinitive Pas(Per1 Sg) : መሠበር
Infinitive Pas(Per1 Pl) : መሠበር
Infinitive Pas(Per2 Sg Masc) : መሠበር
Infinitive Pas(Per2 Sg Fem) : መሠበር
Infinitive Pas(Per2 Pl Masc) : መሠበር
Infinitive Pas(Per2 Pl Fem) : መሠበር
Infinitive Pas(Per3 Sg Masc) : መሠበር
Infinitive Pas(Per3 Sg Fem) : መሠበር
Infinitive Pas(Per3 Pl Masc) : መሠበር
Infinitive Pas(Per3 Pl Fem) : መሠበር
Parti Act (Per1 Sg) : ሠባሪ
Parti Act (Per1 Pl) : ሠባሪ
Parti Act (Per2 Sg Masc) : ሠባሪ
Parti Act (Per2 Sg Fem) : ሠባሪ
Parti Act (Per2 Pl Masc) : ሠባሪ
Parti Act (Per2 Pl Fem) : ሠባሪ
Parti Act (Per3 Sg Masc) : ሠባሪ
Parti Act (Per3 Sg Fem) : ሠባሪ
Parti Act (Per3 Pl Masc) : ሠባሪ
Parti Act (Per3 Pl Fem) : ሠባሪ
Parti Pas(Per1 Sg) : ተሠባሪ
Parti Pas(Per1 Pl) : ተሠባሪ
Parti Pas(Per2 Sg Masc) : ተሠባሪ
Parti Pas(Per2 Sg Fem) : ተሠባሪ
Parti Pas(Per2 Pl Masc) : ተሠባሪ
Parti Pas(Per2 Pl Fem) : ተሠባሪ
Parti Pas(Per3 Sg Masc) : ተሠባሪ
Parti Pas(Per3 Sg Fem) : ተሠባሪ
Parti Pas(Per3 Pl Masc) : ተሠባሪ
Parti Pas(Per3 Pl Fem) : ተሠባሪ
CompPerf Act (Per1 Sg) : ሠብሬያለሁ
CompPerf Act (Per1 Pl) : ሠብረናል
CompPerf Act (Per2 Sg Masc) : ሠብረካል
CompPerf Act (Per2 Sg Fem) : ሠብረሻል
CompPerf Act (Per2 Pl Masc) : ሠብራችኋል
CompPerf Act (Per2 Pl Fem) : ሠብራችኋል
CompPerf Act (Per3 Sg Masc) : ሠበሯል
CompPerf Act (Per3 Sg Fem) : ሠብራለች
CompPerf Act (Per3 Pl Masc) : ሠብረዋል
CompPerf Act (Per3 Pl Fem) : ሠብረዋል
CompPerf Pas(Per1 Sg) : ተሠብሬያለሁ
CompPerf Pas(Per1 Pl) : ተሠብረናል
CompPerf Pas(Per2 Sg Masc) : ተሠብረካል
CompPerf Pas(Per2 Sg Fem) : ተሠብረሻል
CompPerf Pas(Per2 Pl Masc) : ተሠብራችኋል
CompPerf Pas(Per2 Pl Fem) : ተሠብራችኋል
CompPerf Pas(Per3 Sg Masc) : ተሠበሯል
CompPerf Pas(Per3 Sg Fem) : ተሠብራለች
CompPerf Pas(Per3 Pl Masc) : ተሠብረዋል
CompPerf Pas(Per3 Pl Fem) : ተሠብረዋል
Cont Act (Per1 Sg) : ዕሠብር
Cont Act (Per1 Pl) : ዕንሠብር
Cont Act (Per2 Sg Masc) : ትሠብር
Cont Act (Per2 Sg Fem) : ትሠብር
Cont Act (Per2 Pl Masc) : ትሠብሩ
Cont Act (Per2 Pl Fem) : ትሠብሩ
Cont Act (Per3 Sg Masc) : ይሠብር
Cont Act (Per3 Sg Fem) : ትሠብር
Cont Act (Per3 Pl Masc) : ይሠብሩ
Cont Act (Per3 Pl Fem) : ይሠብሩ
Cont Pas(Per1 Sg) : ዕሠበር
Cont Pas(Per1 Pl) : ዕንሠበር
Cont Pas(Per2 Sg Masc) : ትሠበር
Cont Pas(Per2 Sg Fem) : ትሠበር
Cont Pas(Per2 Pl Masc) : ትሠበሩ
Cont Pas(Per2 Pl Fem) : ትሠበሩ
Cont Pas(Per3 Sg Masc) : ይሠበር
Cont Pas(Per3 Sg Fem) : ትሠበር
Cont Pas(Per3 Pl Masc) : ይሠበሩ
Cont Pas(Per3 Pl Fem) : ይሠበሩ

Appendix- C - The Amharic Fidäl



D - Test Examples

The test examples are generated using the resource grammar library function as shown below and are intended merely for proof-reading. They can also reflect, to some extent, the coverage of the work done on Amharic resource library.

EX1 Lang> p -lang=LangEng "**I will kill him cleanly , cleverly and badly**"|l -treebank -all -to_amharic
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TFut ASimul) PPos (PredVP (UsePron i_Pron) (AdvVP (ComplSlash (SlashV2a kill_V2) (UsePron he_Pron)) (ConjAdv and_Conj (ConsAdv (PositAdvAdj clean_A) (BaseAdv (PositAdvAdj clever_A) (PositAdvAdj bad_A)))))))) NoVoc

LangAmh: እኔ እርሱን በ ንጹህ ሁኔታ ፣ በ ብልህ ሁኔታ እና በ መጥፎ ሁኔታ ዕገድላለሁ

EX2 Lang> p -cat=S -lang=LangEng "**he , John and she will come**"|l -treebank -all -to_amharic
Lang: UseCl (TTAnt TFut ASimul) PPos (PredVP (ConjNP and_Conj (ConsNP (UsePron he_Pron) (BaseNP (UsePN john_PN) (UsePron she_Pron)))) (UseV come_V))

LangAmh: እርሱ ፣ ዮሃንስ እና እርሷ ይመጣሉ

EX3 Lang> p -cat=S -lang=LangEng "**John and I will come to you**"|l -treebank -all -to_amharic
Lang: UseCl (TTAnt TFut ASimul) PPos (PredVP (ConjNP and_Conj (BaseNP (UsePN john_PN) (UsePron i_Pron))) (AdvVP (UseV come_V) (PrepNP to_Prep (UsePron youPl_Pron))))

LangAmh: ዮሃንስ እና እኔ ወደ እናንተ ዕንመጣለን

EX4 Lang> p -cat=S -lang=LangEng "**this long girl is so clever and beautiful**"|l -treebank -all -to_amharic
Lang: UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN (DetQuant this_Quant NumSg) (AdjCN (PositA long_A) (UseN girl_N))) (UseComp (CompAP (AdAP so_AdA (ConjAP and_Conj (BaseAP (PositA clever_A) (PositA beautiful_A)))))))

LangAmh: ይህች ረጅም ልጃገረድ በጣም ብልህ እና ቆንጆ ነች

EX5 Lang> p -cat=S -lang=LangEng "**the long girls are clever , young and beautiful**"|l -treebank -all -to_amharic
Lang: UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumPl) (AdjCN (PositA long_A) (UseN girl_N))) (UseComp (CompAP (ConjAP and_Conj (ConsAP (PositA clever_A) (BaseAP (PositA young_A) (PositA beautiful_A)))))))

LangAmh: ረጅሞቹ ልጃገረዶች ብልሆች ፣ ወጣቶች እና ቆንጆዎች ናቸው

EX6 Lang> p -lang=LangEng "**a boy hit a girl by the car**" | l -lang=LangAmh -to_amharic -treebank

Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PPos (PredVP (DetCN (DetQuant IndefArt NumSg) (UseN boy_N)) (ComplSlash (SlashV2a hit_V2) (AdvNP (DetCN (DetQuant IndefArt NumSg) (UseN girl_N)) (PrepNP by8agent_Prep (DetCN (DetQuant DefArt NumSg) (UseN car_N)))))))) NoVoc

LangAmh: አንድ ልጅ አንዲት ልጃገረድን መኪናዉ ጋ መታ

Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PPos (PredVP (DetCN (DetQuant IndefArt NumSg) (UseN boy_N)) (ComplSlash (SlashV2a hit_V2) (AdvNP (DetCN (DetQuant IndefArt NumSg) (UseN girl_N)) (PrepNP by8means_Prep (DetCN (DetQuant DefArt NumSg) (UseN car_N)))))))) NoVoc

LangAmh: አንድ ልጅ አንዲት ልጃገረድን በ መኪናዉ መታ

EX7 Lang> p -lang=LangEng "**the girl was hit by the car**" | l -lang=LangAmh -to_amharic -treebank
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumSg) (UseN girl_N)) (AdvVP (PassV2 hit_V2) (PrepNP by8agent_Prep (DetCN (DetQuant DefArt NumSg) (UseN car_N)))))) NoVoc

LangAmh: ልጃገረዷ መኪናዉ ጋ ተመታች

Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumSg) (UseN girl_N)) (AdvVP (PassV2 hit_V2) (PrepNP by8means_Prep (DetCN (DetQuant DefArt NumSg) (UseN car_N)))))) NoVoc

LangAmh: ልጃገረዷ በ መኪናዉ ተመታች

EX8 Lang> p -lang=LangEng "**the young , red and beautiful boys come here or you will come there**" | l -lang=LangAmh -to_amharic -treebank
Lang: PhrUtt NoPConj (UttS (ConjS or_Conj (BaseS (UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumPl) (AdjCN (ConjAP and_Conj (ConsAP (PositA young_A) (BaseAP (PositA red_A) (PositA beautiful_A)))) (UseN boy_N))) (AdvVP (UseV come_V) here_Adv))) (UseCl (TTAnt TFut ASimul) PPos (PredVP (UsePron youPl_Pron) (AdvVP (UseV come_V) there7to_Adv)))))) NoVoc

LangAmh: ወጣቶቹ ፣ ቀዮቹ እና ቆንጆዎቹ ልጆች እዚህ ይመጣሉ ወይም እናንተ ወደዚያ ትመጣላችሁ

Lang: PhrUtt NoPConj (UttS (ConjS or_Conj (BaseS (UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumPl) (AdjCN (ConjAP and_Conj (ConsAP (PositA young_A) (BaseAP (PositA red_A) (PositA beautiful_A)))) (UseN boy_N))) (AdvVP (UseV come_V) here_Adv))) (UseCl (TTAnt TFut ASimul) PPos (PredVP (UsePron youPol_Pron) (AdvVP (UseV come_V) there7to_Adv)))))) NoVoc

LangAmh: ወጣቶቹ ፣ ቀዮቹ እና ቆንጆዎቹ ልጆች እዚህ ይመጣሉ ወይም እርስዎ ወደዚያ ይመጣሉ

Lang: PhrUtt NoPConj (UttS (ConjS or_Conj (BaseS (UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN (DetQuant DefArt NumPl) (AdjCN (ConjAP and_Conj (ConsAP (PositA young_A) (BaseAP (PositA red_A) (PositA beautiful_A)))) (UseN boy_N))) (AdvVP (UseV come_V) here_Adv))) (UseCl (TTAnt TFut ASimul) PPos (PredVP (UsePron youSg_Pron) (AdvVP (UseV come_V) there7to_Adv)))))) NoVoc

LangAmh: ወጣቶቹ ፣ ቀዮቹ እና ቆንጆዎቹ ልጆች እዚህ ይመጣሉ ወይም አንተ ወደዚያ ትመጣለህ

EX9 Lang> p -lang=LangEng "**she didn't come to university before the industry**" | l -lang=LangAmh -treebank -to_amharic
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PNeg (PredVP (UsePron she_Pron) (AdvVP (AdvVP (UseV come_V) (PrepNP to_Prep (MassNP (UseN university_N)))) (PrepNP before_Prep (DetCN (DetQuant DefArt NumSg) (UseN industry_N))))))) NoVoc

LangAmh: እርሷ ከ ኢንዱስትሪዉ በፊት ወደ ዩንቨርሲቲ አል &+ መጣች &+ ም

EX10 Lang> p -lang=LangEng "**ash laughs on fire**" | l -lang=LangAmh -treebank -to_amharic
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP (MassNP (UseN ashes_N)) (AdvVP (UseV laugh_V) (PrepNP on_Prep (MassNP (UseN fire_N))))))) NoVoc

LangAmh: አመድ በ እሳት ላይ ይሥቃል

EX11 Lang> p -lang=LangEng "**I bought 3 houses in Paris**"| l -lang=LangAmh -treebank -to_amharic
Lang: PhrUtt NoPConj (UttS (UseCl (TTAnt TPast ASimul) PPos (PredVP (UsePron i_Pron) (AdvVP (ComplSlash (SlashV2a buy_V2) (DetCN (DetQuant IndefArt (NumCard (NumDigits (IDig D_3)))) (UseN house_N))) (PrepNP in_Prep (UsePN paris_PN)))))) NoVoc

LangAmh: እኔ 3 ቤቶችን ፓሪስ ውስጥ ገዛሁ

EX12 Lang> p -lang=LangEng "**does he come today**"| l -lang=LangAmh -treebank -to_amharic
Lang: PhrUtt NoPConj (UttQS (UseQCl (TTAnt TPres ASimul) PPos (QuestCl (PredVP (UsePron he_Pron) (AdvVP (UseV come_V) today_Adv))))) NoVoc

LangAmh: እርሱ ዛሬ ይመጣል እንዴ