Thesis for the degree of Doctor of Philosophy

Computational Linguistics Resources for Indo-Iranian Languages

Shafqat Mumtaz Virk

CHALMERS | GÖTEBORG UNIVERSITY

Department of Computer Science and Engineering
Chalmers University of Technology &
University of Gothenburg
Gothenburg, Sweden 2013

Computational Linguistics Resources for Indo-Iranian Languages
Shafqat Virk

ISBN 978-91-628-8706-3
Technical report Number 96D

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

Printed at Chalmers, Gothenburg, 2013

Abstract

Can computers process human languages? During the last fifty years, two main approaches have been used to find an answer to this question: data-driven (i.e. statistics based) and knowledge-driven (i.e. grammar based). The former relies on the availability of a vast amount of electronic linguistic data and the processing capabilities of modern-age computers, while the latter builds on grammatical rules and classical linguistic theories of language.

In this thesis, we use mainly the second approach and elucidate the development of computational ("*resource*") grammars for six Indo-Iranian languages: Urdu, Hindi, Punjabi, Persian, Sindhi, and Nepali. We explore different lexical and syntactical aspects of these languages and build their *resource grammars* using the Grammatical Framework (GF) – a type theoretical grammar formalism tool.

We also provide computational evidence of the similarities/differences between Hindi and Urdu, and report a mechanical development of a Hindi resource grammar starting from an Urdu resource grammar. We use a *functor* style implementation that makes it possible to share the commonalities between the two languages. Our analysis shows that this sharing is possible upto 94% at the syntax level, whereas at the lexical level Hindi and Urdu differed in 18% of the basic words, in 31% of tourist phrases, and in 92% of school mathematics terms.

Next, we describe the development of wide-coverage morphological lexicons for some of the Indo-Iranian languages. We use existing linguistic data from different resources (i.e. dictionaries and WordNets) to build uni-sense and multi-sense lexicons.

Finally, we demonstrate how we used the reported grammatical and lexical resources to add support for Indo-Iranian languages in a few existing GF application grammars. These include the Phrasebook, the mathematics grammar library, and the Attempto controlled English grammar. Further, we give the experimental results of developing a wide-coverage grammar based arbitrary text translator using these resources. These applications show the importance of such linguistic resources, and open new doors for future research on these languages.

Acknowledgments

First of all, I would like to extend my sincere thanks to my main supervisor Prof. Aarne Ranta, my co-supervisor Prof. K.V.S Prasad and the other members of my PhD committee including Prof. Bengt Nordström and Prof. Claes Strannegård for their continuous advice, support, and encouragement. I started my PhD without any comprehensive knowledge of the field, and practical experience of the tools used in this study. However, I was very lucky to have supervisors who encouraged me more than what I deserved, cared a lot about my work, and promptly answered to all of my questions and queries regarding our work.

I am also very grateful to all of my colleagues including Dinesh Simkhada, Elnaz Abolahrar, Jherna Devi Oad, Krasimir Angelov, Muhammad Humayoun, Olga Caprotti, Thomas Hallgren, and all others for their contributions and useful suggestions to make it possible for me. I would also like to mention that Muhammad Azam Sheikh a PhD student, Prof. Graham Kemp, and particularly Prof. K.V.S Prasad helped me to improve the technical quality of the thesis. I am grateful for their part.

I would like to give a very special acknowledgement and gratitude to my parents for the efforts they made to make me climb so high. I can't forget the nights my mother spent awake for me. The fear that she might not be able to make me wake-up and study, if she goes to the bed herself, kept her awake throughout the nights. I also can't forget the bicycle rides my father gave me to drop me off at school, while teaching me lessons on the way. I can't stop my tears, whenever I remember those days. I am also obliged to my siblings and their families for the prayers, wishes, and encouragement, which played a very vital role to achieve this goal.

I would like to thank my wife for her love and continuous support in my hard times, without that all this was not possible. What to say about my son – *Saad Shafqat Virk* – his smiles, hugs, and naughtiness were simply priceless and must have ingredients to get the thesis ready.

Apart from the technical and moral support, the financial support was equally important to complete this thesis. I would like to acknowledge the Higher Education Commission of Pakistan (HEC), University of Engineering & Technology Lahore Pakistan, the MOLTO Project: European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° FP7-ICT-247914, and Graduate School of Language Technology (GSLT) Gothenburg Sweden for providing me the financial assistance.

Contents

Part I

Preliminaries

Chapter 1

Introduction

In this introductory chapter, we start with a general overview of the field, and continue to give a detailed introduction of the Grammatical Framework (GF). This is followed by a brief description of the Indo-Iranian languages and their computational resources. Major motivations behind this study and a short summary of the main contributions together with the organization of the thesis conclude the chapter. The discussion in this chapter is largely based on the GF book [Ranta, 2011] and other publications on GF including [Ranta, 2004], [Ranta, 2009a], and [Ranta, 2009b].

1.1 Background

The history of language study dates back to Iron Age India, when Yaska (6^{th} c BC) and Pāṇini (4^{th} c BC) made the first recorded attempts to develop systematic grammars (i.e. a set of rules of a language). However, the field of computational linguistics (i.e. using computers to perform language engineering) is very young. It can be traced back to the mid of 1940's, when Donald Booth and D.V.H Britten (1947) produced a detailed code for realizing dictionary translation on a digital computer. Machine Translation was the first computer-based application related to natural language processing (NLP). In the early days of machine translation, it was believed that the differences among languages are only at the levels of vocabulary and word order. This resulted in poor translations produced by the early machine translation systems. These systems were based on a dictionary-lookup approach without considering lexical, syntactic, and semantic ambiguities inherent in languages. In 1957, when Chomsky introduced the idea of generative grammars in his book titled *Syntactic Structures* [Chomsky, 1957], the NLP community got a better insight of the field. Many modern theories, e.g. Relational Grammar [Blake, 1990], Generalized Phrase Structure Grammar (GPSG) [Gazdar et al., 1985], Head Driven Phrase Structure Grammar (HPSG) [Carl and Ivan, 1994], and Lexical Functional Grammar (LFG) [Dalrymple, 2001], find their origin in the generative grammar school of thought. Historically, a number of tools and/or programming languages have been designed to implement these theories practically. Examples include the practical categorical grammar formalism: LexGram [Köning, 1995], a special purpose programming language for grammar writing: NLYACC [Ishii et al., 1994], and Lexical Knowledge Builder system for HPSG [Copestake, 2002]. The work reported in this thesis uses the Grammatical Framework (GF) [Ranta, 2004, Ranta, 2011] as a development tool.

1.2 Grammatical Framework (GF)

GF is a type theoretical grammar formalism, which is based on Martin-Löf's type theory [Martin-Löf, 1982]. Linguistically, GF grammars are close to Montague grammars [Montague, 1974]. In Montague's opinion, there is no important theoretical difference between natural languages and formal languages, such as programming languages, and both can be treated equally. This means that in his view, it is possible to formalize natural languages in the same way as formal languages. GF was started in the early 1990's with the objective to build an integrated formalization of natural language

syntax and semantics [Ranta, 2011]. It can be viewed as a special purpose functional programming language designed for writing natural language grammars and applications [Ranta, 2004]. It combines modern functional-programming concepts (e.g. abstraction and higher order functions) with useful programming-language features (e.g. static type system, module system, and the availability of libraries).

1.2.1 Types of Grammars in GF

Natural languages are highly complex and ambiguous, which makes it very hard to engineer them precisely for computational purposes. There are many low-level morphological and grammatical details, such as inflection, word-order, agreement etc. that need to be considered. This is a hard task, especially for those who do not possess enough expertise both on the linguistic and the computational side. Such complexities cannot be reduced (because they are naturally there), but they can be hidden under the umbrella of software libraries.

Ambiguity next. Consider the sentence 'He went to the bank'. There are ten senses of the word `bank` as a noun in the Princeton WordNet [Miller, 1995]. If not more, there are at least two possible interpretations of the above given sentence (1) either he went to the (bank as a sloping land) or (2) he went to the (bank as a financial institution). In general, ambiguities are very difficult to resolve, but many of the lexical ambiguities can be resolved by domain specificity. For example, if we know that we are in a financial domain, it becomes easy to interpret that most probably he went to the (bank as a financial institution).

GF tries to address the challenges of both complexity and ambiguity by providing two types of grammars: *resource grammars* and *application grammars*.

Resource Grammars

Resource grammars are general-purpose grammars that encode general grammatical rules of a natural language [Ranta, 2009b] at both morphological and syntactical levels. These grammars are supposed to be written by linguists, who know better the grammatical rules (e.g. agreement, word order, etc.) of the language. These grammars are then distributed to application developers, in the form of libraries, who can access them through a common resource grammar API, and use them to develop domain-specific application grammars. This approach assists the application developers and provides a way to deal with the complexities of natural languages.

Application Grammars

Application grammars are domain-specific grammars that encode domain-specific constructions. These grammars are supposed to be written by domain experts, who are familiar with domain terminologies. Since the scope of these grammars is limited to a particular domain, and normally they have clearly defined semantics, it becomes easier to handle the lexical and syntactical ambiguities.

1.2.2 GF Resource Grammar Library

The GF resource grammar library (RGL)[Ranta, 2009b] is a set of parallel resource grammars. It is a key component of GF and currently consists of libraries of 26 natural languages. In principle, RGL is similar to the standard software libraries that are provided with many modern programming languages like C, C++, Java, Haskell, etc. The objective of both is the same, and that is to assist the application developers. Consider the following example to see how the availability of RGL can simplify the task of writing application grammars.

Suppose an application developer wants to build the complex noun `black car` from the adjective `black` and the common noun `car`. One possibility for the developer is to write a function that takes an adjective and a common noun (i.e. `black` and `car` in this case) as inputs and produces a complex noun (i.e. `black car`) as output. The function needs to take care of selecting appropriate inflectional forms of the words. As English adjectives do not inflect for number, gender, etc., selecting appropriate forms may appear to be straightforward (i.e. same form of an adjective is attached to a common noun irrespective of number, gender, and case of the common noun). However, the picture becomes more complicated for the languages with rich morphology like Urdu. In Urdu adjectives inflect for number, gender and case [Shafqat et al., 2010]. So, the function should take care of selecting the appropriate form of an adjective agreeing with number, gender, and case of the common noun. Additionally, other grammatical details such as word order should also be in accordance.

An alternative approach is to encapsulate all such linguistic details in a pre-defined function and provide it as a library function. Later, the application grammar developer can use this function with ease. As an example, with the availability of library functions the above task to build the complex noun can easily be achieved by the following single line of the code:

```
For English:
mkCN (mkA "black") (mkN "car")
```

```
For Urdu:
mkCN (mkA "کالا") (mkN "کار")
```

For English, the API function `mkN` takes the string argument `car` and builds the noun `car`. Similarly, the API function `mkA` builds an adjective from its string argument `black`. Finally, `mkCN` function builds the final adjectival modified complex noun from the adjective `black` and the noun `car`. In this approach, the application developer, only, has to learn how to use the API functions (i.e. `mkCN, mkN, and mkA`), and let these functions deal with the low-level linguistic details. This helps the application developer to concentrate on the problem at hand rather than concentrating on low level linguistic issues.

Historically, GF and its resource library have been used to develop a number of multilingual and/or monolingual application grammars including but not limited to the Phrasebook [Ranta et al., 2012], WebAlt [Caprotti, 2006], GF-Key [Johannisson, 2005]. Even though the idea of providing resource grammars as libraries is new in GF, there exist other resource grammar packages. For example the multilingual resource-grammar package of CLE (Core Language Engine, [Rayner et al., 2000]), Pargram [Butt et al., 2002] and LinGo Matrix [Bender and Flickinger, 2005].

1.2.3 Multilingualism

A distinguishing feature of GF grammars is multilingualism. GF grammars maintain Haskell Curry's distinction between tectogrammatical (abstract) and phenogrammatical (concrete) structures [Curry, 1961]. This makes it possible to have multiple parallel concrete syntaxes for a common abstract syntax, which results in multilingual grammars. The abstract and concrete syntax are two levels of GF grammars explained in the following subsections.

Abstract Syntax

An abstract syntax is a logical representation of a grammar. It is common to a set of languages, and is based on the fact that the same categories (e.g. nouns, verbs, adjectives) and the same syntactical rules (e.g. predication, modification) may appear in many languages [Ranta, 2009b]. This commonality is captured in the abstract syntax, which abstracts away from the complexities (i.e. word order, agreement, etc.) involved in language grammars leaving them to the concrete syntax.

Concrete Syntax

A concrete syntax describes the actual surface form of the common abstract syntax in a particular natural language. It is language dependent, and all the complexities involved in a particular language are handled in this part. This is demonstrated practically in the next section.

1.2.4 A Complete Example

We give a small multilingual grammar for generating remarks like 'tasty food', 'bad service', 'good environment' etc. about a hotel. These kinds of remarks can be found on hotel web-pages and blogs. Even though this example is not grammatically very rich, it is good enough to serve our purposes of showing:

- How the idea of a common abstract syntax and multiple parallel concrete syntaxes works in GF.
- How we can deal with the language specific details in the concrete syntax.
- How the abstract syntax abstracts away from the complexities involved in a language leaving them to the concrete syntax.

Further, it is also important to mention that neither the resource grammars nor the resource grammar library API functions have been used to implement the example grammar. One purpose of building it from scratch is to show how the actual resource grammars have been build.

The grammar has one common abstract syntax and four parallel concrete syntaxs (one for each of English, Urdu, Persian, and Hindi). The abstract syntax is given below:

```
abstract Remarks = {
  cat
    Item, Quality, Remark ;
  fun
    good : Quality ;
    bad : Quality ;
    tasty : Quality ;
    fresh : Quality ;
    food : Item ;
    service : Item ;
```

```
    environment : Item ;
    mkRemark : Quality -> Item -> Remark ;
};
```

The abstract syntax contains a list of categories (declared by the keyword `cat` in the above given GF code) and a list of grammatical functions (declared by the keyword `fun`). In this example, we have three different categories. We name them `Item`, `Quality` and `Remark`. One can say that `Item` and `Quality` are lexical categories, and `Remark` is a syntactical category (as it is grammatically constructed from other categories). Next, the abstract syntax has a list of grammatical functions (e.g. 'good', 'bad', 'tasty'). These functions either declare the words as constants of particular lexical categories, or define how different syntactical categories can be constructed from the lexical categories (e.g. definition of 'mkRemark' in the given code). Next, we give the concrete syntaxes.

English Concrete Syntax

A concrete syntax assigns a linearization type (declared by the keyword `lincat` in the code given below) to each category and a linearization function (declared by the keyword `lin`) to each function.

```
concrete RemarksEng of Remarks = {
  lincat
    Quality, Item, Remark = {s : Str } ;
  lin
    good = {s = "good" } ;
    bad = {s = "bad" } ;
    tasty = {s = "tasty" } ;
    fresh = {s = "fresh" } ;
    food = {s = "food" } ;
    service = {s = "service" } ;
    environment = {s = "environment" } ;
    mkRemark quality item = {s = quality.s ++ item.s } ;
 };
```

The category linearization rule states that all three categories (i.e. `Quality`, `Item`, and `Remark`) are of the record-type (indicated by curly brackets). This record has one field labeled as 's', which is of the string type. The function linearization rules assign the actual surface form to each function. In the above code, each function of the type `Quality` or `Item` simply gets the actual string representation, while `Remarks` are constructed by concatenating the

corresponding constituent strings (see the `mkRemark` function in the above code).

Urdu Concrete Syntax

Here, the picture becomes a bit more complicated because the category `Quality` inflects for `Gender`. So, a simple string type structure is not enough to store all inflectional forms of the category `Quality`. We need a richer structure – such as a table type structure. Consider the following code to see how this is achieved in GF. Note the IPA (International Phonetics Association) representations of the strings are preceded by '- -', which is used to insert comments in the GF code.

```
concrete RemarksUrd of Remarks = {
 flags
  coding = utf8;
 Param Gender = Masc | Fem ;

 lincat
  Quality = {s : Gender => Str} ;
  Item = {s : Str ; g : Gender} ;
  Remark = {s : Str } ;
 lin
  good = { s = table {Masc => " اچھا ";   -- acchaː
                      Fem => "اچھی"}};  -- acchiː
  bad = { s = table {Masc => " برا " ;   -- buraː
                      Fem => "بری" }};   -- buriː
  tasty = { s = table {Masc=> "مزیدار";   -- mazeːdaːr
                         Fem => "مزیدار"}};  -- mazeːdaːr
  fresh = { s = table {Masc => "تازہ" ;   -- taːzaː
                      Fem => "تازہ " }};   -- taːzaː
  food = { s = "کھانا" ; g = Masc } ;    -- kʰaːnaː
  service = { s = "سروس " ; g = Fem } ;   -- sarvis
  environment = {s = "ماحول" ; g = Masc } ;   -- mahoːl

  mkRemark quality item = {s = quality.s ! item.g ++ item.s } ;
       };
```

In the lincat rule for `Quality`, `s` is an object of a table-type structure declared as: `{s : Gender => Str}`. It is read as: "a table from Gender to String", where `Gender` is a parameter defined as follows:

```
param Gender = Masc | Fem ;
```

This structure shows how we formalize inflection tables in GF, which are then used to store different inflectional forms. For example, now we are able to store both masculine and feminine forms of the Quality `good`. The following line from the above given code does this task.

```
good = { s = table {Masc => "اچھا" ;   -- acc^ha:
                    Fem => "اچھی" }} ;  -- acc^hi:
```

Next, the `Item` category has an inherent gender property. So, the `lincat` rule of the `Item` is the following:

```
lincat Item = {s : Str ; g : Gender} ;
```

This record has two fields. `s` is a simple string to store the actual string representation of the `Item`, while `g` is of the type `Gender` and stores the inherent gender information of the `Item`. This information is used to select the appropriate inflectional form of `Quality` from its inflection table, which is in agreement with the gender of an `Item`. This is done in the `mkRemark` function i.e.:

```
mkRemark quality item =
{s = quality.s ! item.g ++ item.s } ;
```

Note, how the gender of the `item` (i.e. item.g) is used to select an appropriate form of the `quality` using the selection operator (`!`). This will ensure the formation of grammatically correct remarks in Urdu. Consider the following example:

اچھا کھانا
acc^ha:_good k^ha:na:_food, good food
اچھی سروس
acc^hi:_good sarvis_service, good service

It is notable that different inflectional forms of the quality `good` are used with the item `food` (which is inherently masculine) and the item `service` (which is inherently feminine). This shows how one can deal with the language-specific agreement features in the concrete syntax. (see Table 1.1 for more examples)

Persian Concrete Syntax

In this concrete syntax, we show how to take care of the word order differences.

```
concrete RemarksPes of Remarks = {
lincat
  Quality, Item, Remark = {s : Str } ;
lin
    bad = {s = "بد"} ;   -- bad
    tasty = {s = "خوشمزه"} ;  -- xošmaza:
    fresh = {s = "تازه"} ;  -- ta:za:
    food = {s = "غذا"} ;  -- Gaza:
    service = {s = "سروس"} ;  -- sarvis
    environment = {s = "محیط"} ;  -- mohe:t

    mkRemark quality item = {s = item.s ++ quality.s };
      };
```

In Persian, the word order is different from Urdu. In Urdu the `quality` preceded the `item` (i.e. an adjective preceded a noun), while in Persian it is the other way around. This can be observed in the following linearization rule:

```
lin mkRemark quality item = {s = item.s ++ quality.s } ;
```

This ensures the correct word order in Persian (see Table 1.1 for examples).

Hindi Concrete Syntax

Finally, we consider the concrete syntax of Hindi. In Hindi the inflection and the word order are very similar to Urdu (at least for this example). The only difference between Urdu and Hindi concrete syntax is the script. Urdu uses Perso-Arabic script while Hindi uses Devanagari script as shown below:

```
concrete RemarksHin of Remarks = {
  Param Gender = Masc | Fem ;
  lincat
   Quality  = {s : Gender => Str} ;
   Item  = {s : Str ; g : Gender} ;
   Remark  = {s : Str } ;
  lin
   good = {s = table {Masc=> "अच्छा" ;  -- acchʰa:
                      Fem => "अच्छी"}};  -- acchʰi:
   bad = {s = table {Masc => "बुरा" ;  -- bura:
                      Fem => "बुरी" }};  -- buri:
   tasty = {s = table {Masc => "स्वादिष्ट" ;  -- sva:diṣṭ
                       Fem=> "स्वादिष्ट"}};  -- sva:diṣṭ
```

```
fresh = {s = table {Masc=>"ताज़ा" ;  -- ta:za:
                    Fem => "ताज़ी"};  -- ta:za:
food  = {s = "खाना" ; g = Masc } ;  -- kʰa:na:
service  = {s = "सेवा" ; g = Fem } ;   -- seva:
environment = {s = "पर्यावरण" ; g = Masc } ; -- parya:varaṇ
mkRemark quality item =
                   {s = quality.s ! item.g ++ item.s};
    };
```

Abstract	English	Hindi	Persian	Urdu
mkRemark fresh food	fresh food	ताज़ा खाना	غذا تازہ	تازہ کھانا
mkRemark bad environment	bad environment	बुरा पर्यावरण	محیط بد	برا ماحول
mkRemark bad service	bad service	बुरी सेवा	سروس بد	بری سروس
mkRemark tasty food	tasty food	स्वादिष्ट खाना	غذا خوشمزہ	مزیدار کھانا

Table 1.1: Multilingual Example Remarks

1.3 Indo-Iranian Languages and their Computational Resources

There exist more than 7000 living natural languages around the world (Ethnologue), which have been genetically classified into 136 different families. Indo-European is one of the top 6 language families with 436 living languages, and around 2.9 billion speakers. This family of languages is further divided into 10 major branches and the Indo-Iranian is the largest branch with 310 languages. Geographically, this branch covers languages spoken in Eastern Europe, Southwest Asia, Central Asia, and South Asia, and has more than one billion native speakers in total. Major languages in this branch are: Hindustani (Hindi and Urdu) – 240 million native speakers, Bengali – 205 million native speakers, Punjabi – 100 million native speakers, and Persian – 60 million native speakers (the numbers are taken from the Wikipedia). There have been a number of individual and combined attempts to build computational resources for these languages. The major work includes:

1. **The PAN Localization**[1] **Project:** a combined project of International Development Research Center (IDRC), Canada and the Center for Research in Urdu Language Processing (CRULP), Pakistan. It involves ten Asian countries including Afghanistan, Bangladesh, Bhutan,

[1] http://www.panl10n.net/

Cambodia, China, Laos, Mongolia, Nepal, Pakistan, and Sri Lanka. Many linguistic resources including fonts, parallel-corpus, keyboard layouts, dictionaries have been developed and released by different partners of this project.

2. **The Indo-WordNet[2] Project:** a project to build a linked WordNet of Indian languages. It started with the Hindi WordNet project, which is based on the ideas from the Princeton WordNet [Miller, 1995], and now has grown to 19 languages with varying size and coverage.

3. **The Hindi/Urdu Treebank[3] Project:** This project has been under construction since 2008. The objective is to build a syntactically, and semantically annotated tree-bank of Hindi/Urdu covering around 400,000 words. Historically, tree-banks (e.g. Penn Treebank[4]) have proved to be very useful linguistic resources that can be used for a number of NLP related tasks including training and testing of parsers.

4. **ParGram Urdu Project[5]:** an on-going project for building a comprehensive Urdu and Hindi grammar using the Lexical Functional Grammar (LFG) framework. It is part of the ParGram[6] project, which aims to build parallel grammars for a number of natural languages including Urdu. However, Urdu is the least implemented language.

5. Being a liturgical (i.e. holy in the religious context) and the oldest language in the region, Sanskrit holds a prominent position in the Indo-Iranian branch, and has influenced strongly the other languages (e.g. Hindi) which evolved around it. Due to a number of reasons, including the complex grammatical structure, it has been of particular interest for both linguistics and computational linguistics community over the years. [Monier-Williams, 1846, Kale, 1894] describes different aspects of the Sanskrit grammar with Pāṇini (4^{th} c BC) being the pioneer one. A toolkit for morphological and phonological processing of Sanskrit was reported in [Huet, 2005]. Many other computational resources including tagger, morphological analyzer, reader and parser for Sanskrit can be found on the Sanskrit Heritage website[7].

[2]http://www.cfilt.iitb.ac.in/indowordnet/
[3]http://verbs.colorado.edu/hindiurdu/index.html
[4]http://www.cis.upenn.edu/~treebank/
[5]http://ling.uni-konstanz.de/pages/home/pargram_urdu/
[6]http://pargram.b.uib.no/
[7]http://sanskrit.inria.fr/index.fr.html

6. A computational grammar for Urdu was reported in [Rizvi, 2007]. This work gives a very detailed analysis of Urdu morphology and syntax. It also describes how to implement the Urdu grammar using the Lexical Functional Grammar (LFG) and the Head-driven Phrase Structure Grammar (HPSG) frameworks.

1.4 Major Motivations

The following are four major motivations behind this study:

1. The GF resource grammar library has support of an increasing number of languages. So far most of these languages belong to the Germanic, Romance, or Slavic branches of the Indo-European family of languages. As mentioned previously, out of the 436 Indo-European languages, 310 languages are Indo-Iranian, which means 70% of the languages in this family belong to the Indo-Iranian branch. Unfortunately, there has not been enough effort in the past to develop computational resources for these languages. One example is the Punjabi language. With around 100 million native speakers, it is the 12th most widely spoken language in the world. When it comes to the computational resources, it is hard to find any grammatical resources for this language. So, the main motivation behind this work is to develop computational resources (grammars and lexicons) of these resource-poor languages (Chapter 2-5).

2. Indo-Iranian languages have some distinctive features like the partial *ergative* behavior of verbs and the *Ezafe*[8] construction. Another motivation behind this work is to explore this dimension, and demonstrate how one can implement such features in GF (Chapter 2 and 4).

3. There are many learned and differing views on whether Hindi and Urdu are one or two languages, but nothing has been proved computationally. Joshi in a news article [Joshi, 2012] supports the slogan 'one language, two scripts', while [Flagship, 2012, Schmidt, 2004, Naim, 1999] give arguments to prove them different at different levels. In this study, we find computational evidence of the similarities/differences between Hindi/Urdu (Chapter 6).

4. Historically, GF and its resource grammar library have been used to develop a number of domain-specific NLP applications, but their use

[8] *Ezafe* is a special grammatical feature of Persian, which is used to link words in phrases [Samvelian, 2007]. It is inherited from Arabic and is commonly used to express noun-adjective linking.

at a wider level is largely unexplored. Recently, there have been some attempts to scale up GF and its resource grammars for open-domain tasks, such as arbitrary text translation. These include the extension of the GF parser with statistical ranking for syntactic disambiguation, and support for robustness [Angelov, 2011]. In this thesis, we take these attempts further by developing wide-coverage lexicons (Chapter 7), and experimenting for a wide-coverage text translator (Chapter 8).

1.5 Main Contributions and the Organization of the Thesis

1.5.1 Grammatical Resources

We started developing an Urdu resource grammar with the major objective to contribute something substantial for the Indo-Iranian languages to the GF resource grammar library. After nine months of work and with approximately 2500 lines of the code, the first version of the Urdu resource grammar was released in the early 2010. The implementation details are given in Chapter 2, which is based on the following workshop paper:

Shafqat M. Virk, M. Humayoun, A. Ranta. *"An Open Source Urdu Resource Grammar"*. Proceedings of the 8th Workshop on Asian Language Resources. In conjunction with COLING 2010.

In this work, I am the major contributor in the development of both morphology and syntax. However, as mentioned in the paper the rules of Urdu morphology are borrowed from a previous work [Humayoun, 2006] on Urdu morphology development.

Hindi is closely related to Urdu, but being able to find contradictory views from the literature on whether Hindi and Urdu are one or two languages, the picture remains mostly unclear. This raised the following research questions:

Is it possible to computationally prove whether Hindi and Urdu are one or two languages? If the languages are different, how much do they differ and at what levels? Can this be measured quantitatively?

To find answers to these research questions, we took the Urdu resource grammar and mechanically developed a Hindi resource grammar using *functors*. Being able to share 94% of the code at the syntax level favors the view that Hindi and Urdu are very similar, but this is true mostly at the syntax level, because at the lexical level, our evaluation results show that Hindi and Urdu

differed in 18% of the basic vocabulary, in 31% of touristic phrases, and in 92% of mathematical terms. The implementation and further experimental details are given in chapter 6 and *Appendix A*. Chapter 6 is based on the following workshop paper:

K.V.S. Prasad and **Shafqat Mumtaz Virk**. *"Computational evidence that Hindi and Urdu share a grammar but not the lexicon"*. In The 3rd Workshop on South and Southeast Asian NLP, COLING 2012.

My main contribution in this work is in the development of the Hindi resource grammar (Prasad helped for linguistic details and Devanagari script) and in adding support for Hindi and Urdu in the Phrasebook and the MGL application grammars. In the writing process, I mainly contributed in Sections 6.2, 6.3 and 6.5.

The lessons we learned from the development of the Urdu and Hindi resource grammar were used to build the Punjabi and the Persian resource grammars. The implementation details are given in chapter 3 and 4 respectively, which are based on the following two conferences papers:

Shafqat Mumtaz Virk and Elnaz Abolahrar. *"An Open Source Persian Computational Grammar"*. Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

In this work my major contribution is in the development of the syntax part. Elnaz is a native Persian speaker, she contributed mostly in the development of the morphology part, and during the testing and the verification processes.

Shafqat M. Virk, M. Humayoun, A. Ranta. *"An Open Source Punjabi Resource Grammar"*. Proceedings of Recent Advances in Natural Language Processing (RANLP), pages 70-76, Hissar, Bulgaria, 12-14 September 2011.

In this work my major contribution is in the development of the syntax part. As it is mentioned in the paper that a Punjabi morphology was developed independently, after a few required adjustments, we have reused the same morphological paradigms in the development of this resource grammar.

Nepali and Sindhi resource grammars were developed as master thesis projects together with Dinesh Simkhada and Jherna Devi Oad respectively. We don't give any implementation details in this thesis, assuming that they can be found in the corresponding thesis reports [Simkhada, 2012] and [Devi, 2012]. However, we include the corresponding language examples and their morphological paradigm documentation in *Appendix B*.

1.5.2 Lexical Resources

Historically, a widely explored and appreciated area of application of the GF resource grammars has been the controlled language implementations. One needs to have comprehensive lexical resources to investigate the possibility of using these resource grammars at wider levels such as open-domain machine translation. In this study, we report the development of comprehensive monolingual and multi-lingual GF lexicons from existing lexical resources such as dictionaries and WordNets. Details are given in Chapter 6.

1.5.3 Applications

At the end, to show the usefulness of these grammatical and lexical resources, we have added support for Urdu and Hindi in a number of controlled languages: the Phrasebook [Ranta et al., 2012], the Mathematical Grammar Library (MGL) [Caprotti and Saludes, 2012], and the Attempto Controlled English (ACE) grammar in GF [Kaljurand and Kuhn, 2013],. Furthermore, we report our experimenting for a grammar based machine translation system using GF resource grammars and wide-coverage lexicons. Details are given in Chapter 7 and 8.

Part II

Grammatical and Lexical Resources

Chapter 2

An Open Source Urdu Resource Grammar

This chapter is based on a workshop paper, and describes the development of an Urdu Resource Grammar. It explores different lexical and grammatical aspect of Urdu, and elucidate how to implement them in GF. It also gives an example to show how the grammar works at different levels: morphology and syntax.

The layout has been changed and the document has been technically improved.

Abstract: In this paper, we report a computational grammar of Urdu developed in the Grammatical Framework (GF). GF is a programming language for developing multilingual natural language processing applications. GF provides a library of resource grammars, which currently supports 16 languages. These grammars follow an Interlingua approach and consist of morphology and syntax modules that cover a wide range of features of a language. We explore different syntactic features of Urdu, and show how to fit them into the multilingual framework of GF. We also discuss how we cover some of the distinguishing features of Urdu such as *ergativity* in verb agreement. The main purpose of the GF resource grammar library is to provide an easy way to write natural language processing applications without knowing the details of syntax and morphology. To demonstrate this, we use the Urdu resource grammar to add support for Urdu in an already existing GF application grammar.

2.1 Introduction

Urdu is an Indo-European language within the Indo-Aryan family, and is widely spoken in South Asia. It is the national language of Pakistan and is one of the official languages of India. It is written in a modified Perso-Arabic script from right-to-left. As regards vocabulary, it has a strong influence of Arabic and Persian along with some borrowings from Turkish and English. Urdu is an SOV language having fairly free word order. It is closely related to Hindi as both originated from a dialect of Delhi region called `khari boli` [Masica, 1991].

We develop a grammar for Urdu, which addresses problems related to automated text translation using an Interlingua approach. It provides a way to precisely translate text, which is described in Section 2.2. Next, we describe different levels of grammar development including morphology (Section 2.3) and syntax (Section 2.4). In Section 2.6, we briefly describe an application grammar which shows how a semantics-driven translation system can be built using these components.

2.2 Grammatical Framework

Grammatical Framework (GF) [Ranta, 2004] can be defined in different ways; one way to put it is that it is a tool for working with grammars. Another way is that it is a programming language for writing grammars, which is based on a mathematical theory about languages and grammars. Many multilingual

dialog and text generation applications have been built using GF and its resource grammar library (see GF homepage[1] for more details).

GF grammars have two levels: *abstract syntax* and *concrete syntax*. The abstract syntax is language independent, and is common to a set of languages in the GF resource grammar library. It is based on common syntactic or semantic constructions, which work for all the involved languages on an appropriate level of abstraction. The concrete syntax, on the other hand, is language dependent and defines a mapping from abstract to actual textual representation in a specific language. GF uses the term 'category' to model different parts of speech (e.g. verbs, nouns, adjectives, etc.). An abstract syntax defines a set of categories, as well as a set of tree building functions. A concrete syntax contains rules telling how these trees are linearized. Separating the tree building rules (abstract syntax) from the linearization rules (concrete syntax) makes it possible to have multiple concrete syntaxes for one abstract. This makes it possible to parse text in one language and translate it to multiple other languages.

Grammars in GF can be roughly classified into two kinds: *resource grammars* and *application grammars*. Resource grammars are general-purpose grammars [Ranta, 2009b] that try to cover the general aspects of a language linguistically, and whose abstract syntax encodes syntactic structures. Application grammars, on the other hand, encode semantic structures, but in order to be accurate they are typically limited to specific domains. They are not written from scratch for each domain, but may use resource grammars as libraries [Ranta, 2009a]. Previously GF has resource grammars for 15 languages: English, Italian, Spanish, French, Catalan, Swedish, Norwegian, Danish, Finish, Russian, Bulgarian, German, Polish, Romanian, and Dutch. Most of these languages are European languages. We have developed resource grammar for Urdu making it the 16th in total and the first South Asian language. Resource grammars for several other languages (e.g. Arabic, Turkish, Persian, Maltese, and Swahili) are under construction.

2.3 Morphology

In every GF resource grammar, a test lexicon of 450 words is provided. The full-form inflection tables are built through special functions called lexical paradigms. The rules for defining Urdu morphology are borrowed from [Humayoun, 2006], which describes the development of Urdu morphology using the Functional Morphology toolkit [Forsberg and Ranta, 2004]. Although it is possible to automatically generate equivalent GF code from it,

[1] www.grammaticalframework.org

we write the rules of morphology from scratch in GF. The purpose is to get better abstractions than are possible in the generated code. Furthermore, we extend this work by including compound words. However, the details of morphology are beyond the scope of this paper, and its focus is on syntax.

2.4 Syntax

While morphology deals with formation and inflection of individual words, syntax tells how these words (parts of speech) are grouped together to build well-formed phrases. In this section, we discuss how this works in Urdu and describe how it is implemented in GF.

2.4.1 Noun Phrases

When nouns are to be used in sentences as part of speech, there are several linguistic details that need to be considered. For example, other words can modify a noun, and nouns may have features such as gender, number, etc. When all such required details are grouped together with a noun, the resulting structure is known as a noun phrase (NP). According to [Butt, 1993], the basic structure of Urdu noun phrase is `(M) H (M)`, where `M` is a modifier and `H` is head of a `NP`. The head word is compulsory, but modifiers may or may not be present. In Urdu modifiers are of two types: pre-modifiers and post-modifiers. The pre-modifiers come before a head noun, for instance, in the adjectival modification (کالی بلی, ka:li: billi:, "black cat") the adjective `black` is a pre-modifier. The post-modifiers come after a head noun, for instance, in the quantification (تم سب , tum sab, "you all") the quantifier `all` is used as a post modifier. In our implementation we represent a NP as follows:

```
   lincat NP : Type = {s : NPCase => Str ; a : Agr} ;

where

param NPCase = NPC Case | NPErg | NPAbl
               |NPIns|NPLoc1NPLoc2
               |NPDat;|NPAcc
param Case = Dir | Obl | Voc ;
param Agr = Ag Gender Number UPerson ;
param Gender = Masc | Fem ;
param UPerson = Pers1| Pers2_Casual
               |Pers2_Familiar | Pers2_Respect
               |Pers3_Near | Pers3_Distant;
```

```
param Number = Sg | Pl ;
```

The curly braces indicates that a `NP` is a record with two fields: `'s'` and `'a'`. `'s'` is an inflection table and stores different forms of a noun phrase. The Urdu `NP` has a system of syntactic cases, which is partly different from the morphological cases of the category noun (N). According to [Butt et al., 2002], the case markers that follow nouns in the form of post-positions cannot be handled at the lexical level through morphological suffixes, and are thus handled at the syntactic level. We create different forms of a noun phrase to handle different case markers. Following is a short description of different cases of a NP:

- NPC Case: this is used to retain the lexical cases of a noun
- NPErg: Ergative case with the case marker 'ne, نے'
- NPAbl: Ablative case with the case marker 'se, سے'
- NPIns: Instrumental case with the case marker 'se, سے'
- NPLoc1: Locative case with the case marker 'mẽ, میں
- NPLoc2: Locative case with the case marker 'par, پر'
- NPDat: Dative case with case the marker 'ko, کو'
- NPAcc: Accusative case with the case marker 'ko, کو'

The second filed is `a:Agr`, which is the agreement feature of a noun phrase. This feature is used for selecting an appropriate form of other categories that agree with nouns. A noun is converted to an intermediate category (i.e. complex noun `CN`; also known as `N-Bar`), which is then converted to a NP category. A `CN` deals with nouns and their modifiers. As an example consider the following adjectival modification:

```
fun AdjCN : AP -> CN -> CN ;

lin AdjCN ap cn = {
     s = \\n,c =>
         ap.s ! n ! cn.g ! c ! Posit ++ cn.s ! n ! c ;
     g = cn.g
     } ;
```

The linearization of `AdjCN` gives us complex nouns such as (ٹھنڈا پانی , ṭʰaṇḍa: pa:ni:, "cold water"), where a CN (پانی ,pa:ni:, "water") is modified by an

AP (ٹھنڈا ,ṭʰaṇḍa:, “cold”). Since Urdu adjectives also inflect for number, gender, case and degree, we need to concatenate an appropriate form of an adjective that agrees with the common noun. This is ensured by selecting the appropriate form of an adjective and a common noun from their inflection tables, using the selection operator (`!`). Since a `CN` does not inflect in degree but the adjective does, we fix the degree to be positive (Posit) in this construction. Other modifiers include possibly adverbs, relative clauses, and appositional attributes.

A `CN` can be converted to a `NP` using different functions. The following are some of the functions that can be used for the construction of a NP.

```
fun DetCN : Det -> CN -> NP (e.g. the boy)
fun UsePN : PN -> NP (e.g. John)
fun UsePron : Pron -> NP (e.g. he)
fun MassNP : CN -> NP (e.g. milk)
```

Different ways of building a NP, which are common in different languages, are defined in the abstract syntax of a resource grammar, but the linearization of these functions is language dependent and is therefore defined in the concrete syntax.

2.4.2 Verb Phrases

A verb phrase is a single or a group of words that acts as a predicate. In our construction an Urdu verb phrase has the following structure:

```
lincat VP = {
       s : VPHForm => {fin, inf: Str} ;
      obj : {s : Str ; a : Agr} ;
      vType : VType ;
      comp : Agr => Str;
      embComp : Str ;
      ad : Str } ;
where

param VPHForm =
        VPTense VPPTense Agr
        |VPReq HLevel
        |VPStem

and

 param VPPTense = VPPres |VPPast |VPFutr;
```

```
param HLevel = Tu |Tum |Ap |Neutr
param Agr = Ag Gender Number UPerson
```

In GF representation a `VP` is a record with different fields. A brief description of these fields follows:

- The most important field is `s`, which is an inflectional table and stores different forms of a verb. It is defined as `s : VPHForm => {fin, inf: Str};` and is interpreted as an inflection table from `VPHForm` to a tuple of two strings (i.e. {fin,inf:Str}). The parameter `VPHForm` has the following three constructors:

  ```
  VPTense VPPTense Agr
  |VPReq HLevel
  |VPStem
  ```

  The constructor `VPTense` is used to store different forms of a verb required to implement the Urdu tense system. At `VP` level, we define Urdu tenses by using a simplified tense system. It has only three tenses, labeled as `VPPres, VPPast, VPFutr` and defined by the parameter `VPPTense`. For every possible combination of the values of `VPPTense` (i.e. `VPPres, VPPast, VPFutr`) and `Agr` (i.e. Gender, Number, UPerson) a tuple of two string values (i.e. `{fin, inf : Str}`) is created. `fin` stores the copula (auxiliary verb), and `inf` stores the corresponding form of a verb.

  The resource grammar has a common API, which has a much-simplified tense system close to that of the Germanic languages. It is divided into tense and anteriority. There are only four tenses named as present, past, future and conditional, and two possibilities of anteriority (Simul and Anter). This means that it allows 8 combinations. This abstract tense system does not cover all the tenses of Urdu, which is structured around tense, aspect, and mood. We have covered the rest of the Urdu tenses at the clause level. Even though these tenses are not accessible by the common API, they can be used in language specific modules.

  The constructor `VPReq` is used to store request forms of a verb. There are four levels of requests in Urdu. Three of them correspond to (تو tu:, تم tum, and آپ a:p) honor levels and the fourth is neutral with respect to honorific level. Finally, the constructor `VPStem` stores the root form of a verb.

  The forms constructed at the VP level are used to cover the Urdu tense system at the clause level. In our implementation, handling tenses at

the clause level rather than at the verb phrase level simplified the VP structure and resulted in a more efficient grammar.

- `obj` is used to store the object of a verb together with its agreement information.

- `vType` field is used to store information about the type of a verb. In Urdu a verb can be transitive, intransitive or di-transitive [Schmidt, 1999]. This information is important, when dealing with ergativity in verb agreement.

- `comp` and `embComp` are used to store complement of a verb. In Urdu the complement of a verb precedes the actual verb. For example, in the sentence (وہ دوڑنا چاہتی ہے, vo: do:ṛna: ca:hti: hæ, "she wants to run"), the verb (دوڑنا, do:ṛna:, "run") is complement of the verb (چاہنا , ca:hna:, "want"). However, in cases where a sentence or a question sentence is the complement of a verb, the complement comes at the very end of a clause. An example is the sentence (وہ کہتا ہے کہ وہ دوڑتی ہے, vo: kehta: hæ ke vo: do:ṛti: hæ, "he says that she runs"). We have two different fields labled `compl` and `embCompl` in the VP structure to deal with these situations.

- `ad` is used to store an adverb. It is a simple string that can be attached to a verb to build a modified verb.

A distinguishing feature of Urdu verb agreement is `ergativity`. Urdu is one of those languages that show split ergativity. The final verb agreement is with direct subject except in the transitive perfective aspect. In that case the verb agreement is with the direct object and the subject takes the ergative case.

In Urdu, verb shows ergative behavior in the case of the simple past tense, but in the case of other perfective aspects (e.g. immediate past, remote past etc.) there are two different approaches. In the first approach the auxiliary verb (cuka: چکا) is used to make clauses. If (cuka: چکا) is used, the verb does not show ergative behavior and the final verb agreement is with direct subjective. Consider the following example:

لڑکا کتاب خرید چکا ہے

laṛka:$_{\_Direct}$ kita:b$_{_Direct}$ xari:d$_{_Root}$ cuka:$_{\_auxVerb}$ hæ$_{_copula}$
The boy has bought a book

The second way to make the clause is.

لڑکے نے کتاب خریدی ہے

laṛke: ne$_{_Erg}$ kita:b$_{_Direct_Fem}$ xari:di:$_{\_Direct\_Fem}$ hæ$_{_copula}$
The boy has bought a book

In the first approach the subject (لڑکا, laṛka:, “boy”) is in the direct case and the auxiliary verb (چکا,cuka:) agrees with the subject, but in the second approach the verb is in agreement with the object and the ergative case of subject is used. However, in the current implementation we follow the first approach.

In the concrete syntax we ensure the ergative behavior with the following code:

```
case vt of {
   VPPast => case vp.vType of {
     (Vtrans| VTransPost) => <NPErg, vp.obj.a>
      _ => <NPC Dir, np.a>
                            } ;
   _ => <NPC Dir, np.a>
    } ;
```

As shown above, in the case of simple past tense if the verb is transitive then the ergative case of a noun is used and agreement is with the direct object. In all other cases, the direct case of a noun is used and the agreement is with the subject.

Next, we describe how a `VP` is constructed at the syntax level. There are different ways, the simplest is:

```
fun UseV : V -> VP ;
```

Where `V` is a morphological category and `VP` is a syntactic category. There are other ways to make a `VP` from other categories. For example:

```
fun AdvVP : VP -> Adv -> VP ;
```

An adverb can be attached to a `VP` to make an adverbial modified `VP`. For example (یہاں سونا, yahã so:na:, ”sleep here”)

2.4.3 Adjective Phrases

At the syntax level, the morphological adjective (i.e. `A`) is converted to a much richer category: adjectival phrase `AP`. The simplest function for this conversion is:

```
fun PositA : A -> AP ;
```

Its linearization is very simple, because the linearization type of the category `AP` is similar to the linearization type of `A`.

```
lin PositA a = a ;
```

There are other ways of making an AP for example:

```
fun ComparA : A -> NP -> AP ;
```

When a comparative AP is created from an adjective and a NP, constant "سے, se" is used between oblique form of a noun and an adjective. For example linearization of the above function follows:

```
lin ComparA a np = {
    s = \\n,g,c,d => np.s ! NPC Obl ++ "سے"
             ++ a.s ! n ! g ! c ! d ;
      } ;
```

2.4.4 Clauses

A clause is a syntactic category that has a variable tense, polarity and order. Predication of a NP and a VP gives the simplest clause.

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

Where a clause is of the following type.

```
lincat Clause = {s : VPHTense => Polarity => Order => Str};
```

The parameter VPHTense has different values corresponding to different tenses in Urdu. The values of this parameter are given below:

```
param VPHTense = VPGenPres | VPPastSimple
               | VPFut | VPContPres
               | VPContPast | VPContFut
               | VPPerfPres | VPPerfPast
               | VPPerfFut | VPPerfPresCont
               | VPPerfPastCont
               | VPPerfFutCont | VPSubj
```

As mentioned previously, the current abstract level of the common API does not cover all tenses of Urdu, we cover them at the clause level and they can be accessed through a language specific module.

The parameter Polarity is used to make positive and negative sentences and the parameter Order is used to make simple and interrogative sentences. These parameters are declared as given below.

```
param Polarity = Pos | Neg
param Order = ODir | OQuest
```

PredVP function will create clauses with variable tense, polarity and order, which are fixed at the sentence level by different functions, one is:

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

Here, `Temp` is a syntactic category, which is in the form of a record having fields for `Tense` and `Anteriority`. `Tense` in the `Temp` category refers to abstract level `Tense` and we just map it to Urdu tenses by selecting the appropriate clause. This will create simple declarative sentence, other forms of sentences (e.g. question sentences) are handled in the corresponding category modules.

2.4.5 Question Clauses and Question Sentences

The resource grammar common API provides different ways to create question clauses. The simplest way is to create it from a simple clause.

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

In Urdu simple interrogative sentences are created by just adding (کیا, kya:, "what") at the start of a direct clause that already has been created at the clause level. Hence, the linearization of above function simply selects the appropriate form of a clause and adds کیا, kya:, "what" at the start. This clause still has variable tense and polarity, which is fixed at the sentence level through different functions, one is:

```
fun UseQCl : Temp -> Pol -> QCl -> QS ;
```

Other forms of question clauses include clauses made with interrogative pronouns `IP`, interrogative adverbs `IAdv`, and interrogative determiners `IDet`. They are constructed through different functions. A couple of them are given below:

```
fun QuestVP : IP -> VP -> QCl (e.g. who walks?)
fun QuestIAdv : IAdv -> Cl -> QCl (e.g. why does he walk?)
```

`IP, IAdv, IDet` are built at morphological level and can also be created with the following functions.

```
fun AdvIP : IP -> Adv -> IP
fun IdetQuant : IQuant -> Num -> IDet ;
fun PrepIP : Prep -> IP -> IAdv ;
```

2.5 An Example

Consider the translation of the sentence "he drinks hot milk" from English to Urdu to see how our proposed system works at different levels. Figure 2.1 shows an automatically generated parse tree for this sentence. As a

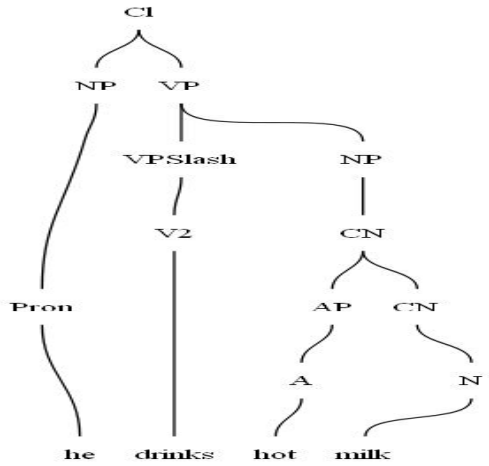

Figure 2.1: Parse Tree

resource grammar developer our goal is to provide correct concrete level linearization of this tree for Urdu. The nodes in this tree represent different categories and its branching shows how a particular category is built from other categories and/or leaves (words from the lexicon). In GF notation these are the syntactic rules, which are declared at the abstract level.

First, consider the construction of the noun phrase 'hot milk' from the lexical units 'hot' and 'milk'. At the morphological level, these lexical units are declared as constants of the lexical category `A` (i.e. adjective) and `N` (i.e. noun) respectively. The following lexical insertion rules covert these lexical constants to the syntactical categories: `AP` (i.e. adjective phrase) and `CN` (i.e. common noun).

```
fun UseA : A -> AP ;
fun UseN : N -> CN ;
```

The resulting `AP` (i.e. 'hot') and `CN` (i.e. 'milk') are passed as inputs to the following function that produces the modified complex noun 'hot milk' as output.

```
fun AdjCN : AP -> CN -> CN ;
```

Finally this complex noun is converted to the syntactic category `NP` through the following function:

```
fun MassNP : CN -> NP ;
```

A correct implementation of these rule in Urdu concrete syntax ensures the correct formation of the noun phrase (گرم دودھ, garam du:d^{h},"hot milk") from

the noun (دودھ, du:d^h, "milk") and the adjective (گرم , garam, "hot").

Similarly, other constituents of the example sentence are constructed individually, and finally the clause (وہ گرم دودھ پیتا ہے, vo: garam du:d^h pi:ta: hæ, "he drinks hot milk") is built from the `NP` (وہ, vo:, "he") and the `VP` (گرم دودھ پیتا ہے, garam du:d^h pi:ta: hæ, "drinks hot milk")

The morphology makes sure that correct forms of words are built during the lexicon development, while language dependent concrete syntax assures that correct forms of words are selected from lexicon and the word order is according to the rules of that specific language.

2.6 An application: Attempto

An experiment of implementing controlled languages in GF is reported in [Ranta and Angelov, 2010]. In this experiment, a grammar for Attempto Controlled English [Attempto, 2008] was implemented using the GF resource library, and then was ported to six languages (English, Finnish, French, German, Italian, and Swedish). To demonstrate the usefulness of our grammar and to check its correctness, we have added Urdu to this set. Now, we can translate Attempto documents between all of these seven languages. The implementation followed the general recipe for how new languages can be added [Angelov and Ranta, 2009] and created no surprises. The details of this implementation are beyond the scope of this paper.

2.7 Related Work

A suite of Urdu resources was reported in [Humayoun, 2006] including a fairly complete open-source Urdu morphology and a small fragment of syntax in GF. In this sense, it is a predecessor of the Urdu resource grammar implemented in a different but related formalism. Like the GF resource library, the ParGram project [Butt and King, 2007] aims at building a set of parallel grammars including Urdu. The grammars in ParGram are connected to each other by transfer functions, rather than a common representation. Further, the Urdu grammar is still the least implemented grammar at the moment.

Other than ParGram, most other work is based on LFG and translation is unidirectional i.e. from English to Urdu only. For instance, the English to Urdu MT System is developed under the Urdu Localization Project [Hussain, 2004, Sarfraz and Naseem, 2007, Khalid et al., 2009]. Zafar and Masood [Zafar and Masood, 2009] reports another English-Urdu MT system developed with the example based approach. [Sinha and Mahesh, 2009]

presents a strategy for deriving Urdu sentences from English-Hindi MT system, which seems to be a partial solution to the problem.

2.8 Future Work

The common resource grammar API does not cover all the aspects of Urdu language, and non-generalizable language-specific features are supposed to be handled in language-specific modules. In our current implementation of Urdu resource grammar we have not covered those features. For example in Urdu it is possible to build a `VP` from only `VPSlash` the (`VPSlash` category represents object missing `VP`) e.g. (کھاتا ہے , k^ha:ta: hæ) without adding the object. This rule is not present in the common API. One direction for future work is to cover such language specific features.

Another direction for future work could be to include the causative forms of a verb, which are not included in the current implementation due to the efficiency issues.

2.9 Conclusion

The resource grammar we developed consists of 44 categories and 190 functions, which cover a fair enough part of the language and are enough for building domain specific application grammars. Since a common API for multiple languages is provided, this grammar is useful in applications where we need to parse and translate the text from one to many other languages.

However, our approach of common abstract syntax has its limitations and does not cover all aspects of Urdu language. This is one reason why it is not possible to use our grammar for arbitrary text parsing and generation.

Chapter 3

An Open Source Punjabi Resource Grammar

The development of the Punjabi resource grammar is described in this chapter, which is based on a conference paper.

The layout has been changed and the document has been technically improved.

Abstract: We describe an open source computational grammar for Punjabi; a resource-poor language. The grammar is developed in GF (Grammatical framework), which is a tool for multilingual grammar formalism. First, we explore different syntactic features of Punjabi and then we implement them in accordance with GF grammar requirements to make Punjabi the 17th language in the GF resource grammar library.

3.1 Introduction

Grammatical Framework [Ranta, 2004] is a special-purpose programming language for multilingual grammar applications. It can be used to write multilingual resource or application grammars (two types of grammars in GF). Multilingualism of the GF grammars is based on the principle that the same grammatical categories (e.g. noun phrases, verb phrases) and the same syntax rules (e.g. predication, modification) can appear in different languages [Ranta, 2009b]. A collection of all such categories and rules, which are independent of any language, makes the abstract syntax of GF resource grammars (every GF grammar has two levels: abstract and concrete). More precisely, the abstract syntax defines semantic conditions to form abstract syntax trees. For example the rule that a common noun can be modified by an adjective is independent of any language and hence is defined in the abstract syntax e.g.:

```
fun AdjCN : AP →CN →CN -- very big blue house
```

However, the way this rule is implemented may vary from one language to another; as each language may have different word order and/or agreement rules. For this purpose, we have the concrete syntax, which is a set of linguistic objects (strings, inflection tables, records) providing rendering and parsing. We may have multiple parallel concrete syntaxes of one abstract syntax, which makes the GF grammars multilingual. Also, as each concrete syntax is independent from others, it becomes possible to model the rules accordingly (i.e. word order, word forms and agreement features are chosen according to language requirements).

Current state-of-the-art machine translation systems such as Systran, Google Translate, etc. provide huge coverage but sacrifice precision and accuracy of translations. On the contrary, domain-specific or controlled multilingual grammar based translation systems can provide a higher translation quality, at the expense of limited coverage. In GF, such controlled grammars are called application grammars.

Writing application grammars from scratch can be very expensive in terms of time, effort, expertise, and money. GF provides a library called the GF resource library that can ease this task. It is a collection of linguistic oriented but general-purpose resource grammars, which try to cover the general aspects of different languages [Ranta, 2009b]. Instead of writing application grammars from scratch for different domains, one may use resource grammars as libraries [Ranta, 2009a] . This method enables him to create the application grammar much faster with a very limited linguistic knowledge. The number of languages covered by GF resource library is growing (17 including Punjabi). Previously, GF and/or its libraries have been used to develop a number of multilingual as well as monolingual domain-specific application grammars, including but not limited to Phrasebook , GF-KeY , and WebALT (see GF homepage[1] for more details).

In this paper we describe the resource grammar development for Punjabi. Punjabi is an Indo-Aryan language widely spoken in Punjab regions of Pakistan and India. Punjabi is among one of the morphologically rich languages (others include Urdu, Hindi, Finish, etc.) with SOV word order, partial ergative behavior, and verb compounding. In Pakistan it is written in Shahmukhi and in India it is written in Gurmukhi script [Humayoun and Ranta, 2010]. Language resources for Punjabi are very limited (especially for the one spoken in Pakistan). With the best of our knowledge this work is the first attempt of implementing a computational Punjabi grammar as open-source software, covering a fair enough part of Punjabi morphology and syntax.

3.2 Morphology

Every grammar in the GF resource grammar library has a test lexicon, which is built through the lexical functions called the lexical paradigms; see [Bringert et al., 2011] for synopsis. These paradigms take lemma of a word and make finite inflection tables, containing the different forms of the word. These words are build according to the lexical rules of that particular language. A suite of Punjabi resources including morphology and a big lexicon was reported by [Humayoun and Ranta, 2010]. With minor required adjustments, we have reused morphology and a subset of that lexicon, as a test lexicon of about 450 words for our grammar implementation. However, the morphological details are beyond the scope of this paper and we refer to [Humayoun and Ranta, 2010] for more details on Punjabi morphology.

[1] www.grammaticalframework.org

3.3 Syntax

While morphology is about types and formation of individual words (lexical categories), it is the syntax, which decides how these words are grouped together to make well-formed sentences. For this purpose, individual words, which belong to different lexical categories, are converted into richer syntactic categories, i.e. noun phrases (NP), verb phrases `(VP)`, and adjectival phrases `(AP)`, etc. With this up-cast the linguistic features such as word-forms, number & gender information, and agreements, etc., travel from individual words to the richer categories. In this section, we explain this conversion from lexical to syntactic categories and afterwards we demonstrate how to glue the individual pieces to make clauses, which then can be used to make well-formed sentences in Punjabi. The following subsections explain various types of phrases.

3.3.1 Noun Phrases

A noun phrase (NP) is a single word or a group of words that does not have a subject and a predicate of its own, and does the work of a noun [Verma, 1974]. First, we show the structure of a noun phrase in our implementation, followed by the description of its different parts.

Structure: In GF, we represent a NP as a record with three fields, labeled as: `'s'` , `'a'` and `'isPron'`:

```
NP: Type = { s : NPCase => Str ;
             a : Agr ;
             isPron : Bool } ;
```

The label `'s'` is an inflection table from NPCase to string `(NPCase => Str)`. `NPCase` has two constructs `(NPC Case, and NPErg)` as shown below:

```
param NPCase = NPC Case | NPErg ;
param Case = Dir | Obl | Voc | Abl ;
```

The construct (`NPC Case`) stores the lexical cases (i.e. direct, oblique, vocative and ablative) of a noun . As an example consider the following table for the noun "boy":

```
s . NPC Dir => مُنڈا --munḍa:
s . NPC Obl => مُنڈے  --munḍe:
s . NPC Voc => مُنڈیا  --munḍi:a:
s . NPC Abl => مُنڈیوں --munḍeõ
```

Other than storing the lexical cases of a noun as shown in the above table, we also construct the ergative case (i.e. NPErg in the code above). We do it at the noun phrase level for the following reason: In Urdu, the case markers that follow a noun in the form of post-positions cannot be handled at lexical level through morphological suffixes and thus need to be handled at syntax level [Butt et al., 2002] . It also applies to Punjabi. So, we construct the ergative case of a noun by attaching the ergative case marker 'ne' to the oblique case of a noun at NP level. For instance, the ergative form of our running example "boy" is:

```
s . NPErg =>  مُنڈے نے munḍe ne_Erg
```

It is used as subjects of perfective transitive verbs (see Section 3.3.5 for more details). The label 'a' represents the agreement feature (Agr) and stores information about gender, number and person that will be used for agreement with other constituents. It is defined as follows:

```
param Agr  = Ag Gender Number Person ;
```

In Punjabi, the gender can be masculine or feminine; number can be singular and plural; and person can be first, second casual, second with respect and third person near & far. These are defined as shown below:

```
param Gender = Masc | Fem ;
param Number = Sg | Pl ;
param Person = Pers1 | Pers2_Casual | Pers2_Respect
             | Pers3_Near | Pers3_Far ;
```

Finally, the label 'isPron' is a Boolean parameter, which shows whether a NP is constructed from a pronoun. This information is important when dealing with the exceptions in ergative behavior of verbs for the first and second person pronouns in Punjabi. For example consider the following constructions:

```
mẽ_I ro:ṭi:_bread kʰa:di:_ate  میں روٹی کھادی
I ate bread.

tũ_You ro:ṭi:_bread kʰa:d i:_ate  تُوں روٹی کھادی
You ate bread.

au: ne_He ro:ṭi:_bread kʰa:di:_ate  اونےروٹی کھادی
He ate bread.

munḍe:_boy ne_ErgMarker ro:ṭi:_bread kʰa:di:_ate  مُنڈے نے روٹی کھادی
The boy ate bread.
```

From the above examples, we can see that, when we have the first or second person pronoun as subject, the ergative case marker is not used (first two examples). However, it is used in all other cases. So for our running example, i.e. the noun (boy, munḍa:), the label `(isPron)` is false.

Construction: First, the lexical category noun `(N)` is converted to an intermediate category, common noun `(CN)` through the (UseN) function.

```
fun UseN : N →CN ; --  مُنڈا munḍa:
```

Then, the common noun is converted to the syntactic category, noun phrase `(NP)`. Three main types of noun phrases are: (1) common nouns with determiners, (2) proper names, and (3) pronouns. We build these noun phrases through different noun phrase construction functions depending on the constituents of a `NP`. As an example consider (1). We define it with a function `DetCN` given below:

```
Every boy, har_every munḍa:_boy
fun DetCN : Det →CN →NP ;
```

Here `(Det)` is a lexical category representing determiners. The above given function takes the determiner `(Det)` and the common noun `(CN)` as parameters and builds the NP, by combining appropriate forms of a determiner and a common noun agreeing with each other. For example if 'every' and 'boy' are the parameters for the above given function the result will be the NP: every boy, har munḍa:. Consider the linearization of DetCN:

```
lin DetCN det cn = {
      s = \\c => detcn2NP det cn c det.n;
      a = agrP3 cn.gdet.n ;
      isPron = False } ;
```

As we know from the structure of a `NP` (given in the beginning of §3.3.1) `'s'` represents the inflection table used to store different forms of a NP built by the following line from the above code:

```
s = \\c => detcn2NP det cn c det.n;
```

Notice that the operator `('\\')` is used as a shorthand to represent different rows of the inflection table `'s'`. An alternative but a verbose code segment for the above line will be:

```
s = table {
        NPC Dir => detcn2NP det cn Dir det.n;
        NPC Obl => detcn2NP det cn Obl det.n;
        NPC Voc => detcn2NP det cn Voc det.n;
```

```
        NPC Abl => detcn2NP det cn Abl det.n
      }
```

Where the helper function detcn2NP is defined as:

```
detcn2NP : Determiner →CN →NPCase →
                  Number →Str =
\dt,cn,npc,n →case npc of {
        NPC c => dt.s ++ cn.s!n!c ;
        NPErg => dt.s ++ cn.s!n!Obl ++ "ne:" } ;
```

Also notice that the selection operator (the exclamation sign `!`) is used to select appropriate forms from the inflection tables (i.e. `cn.s!n!c`, which means the form of the common noun with number `'n'` and case `'c'` from the inflection table `cn.s`). Other main types of noun phrases (2) and (3) are constructed through the following functions.

```
fun UsePN : PN →NP ; Ali, eli:
fun UsePron : Pron →NP ; he, oo
```

This covers only three main types of noun phrases, but there are other types of noun phrases as well, i.e. adverbial post-modified NP, adjectival modified common nouns etc. In order to cover them, we have one function for each such construction. Few of these are given below; for full details we refer to [Bringert et al., 2011].

```
Paris today, ajj_today pi:ras_Paris
fun AdvNP : NP →Adv →NP ;

Big house, vaḍa:_big gʰar_house
fun AdjCN : AP →CN →CN ;
```

3.3.2 Verb Phrases

A verb phrase (`VP`), as a syntactic category, is the most complex structure in our constructions. It carries the main verb and auxiliaries (such as adverb, object of the verb, type of the verb, agreement information, etc.), which are then used in the construction of other categories and/or clauses.

Structure: In GF, we represent a verb phrase as a record, as shown below:

```
VPH : Type = {
s  : VPHForm => {fin, inf : Str} ;
obj  : {s : Str ; a : Agr} ;
vType  : VType ;
```

```
comp  : Agr =>Str;
ad    : Str ;
embComp : Str} ;
```

The label 's' represents an inflection table, which keeps a record with two string values, i.e. `fin, inf : Str` for every value of the parameter VPH-Form, which is defined as shown below:

```
param VPHForm = VPTense VPPTense Agr | VPInf | VPStem ;
param VPPTense = PPres | VPPast | VPFutr | VPPerf;
```

The structure of `VPHForm` makes sure that we preserve all inflectional forms of the verb. In it we have three cases: (1) Inflectional forms inflecting for tense `(VPPTense)` and `number, gender, person`. (2) The second constructor `(VPInf)` carries the infinitive form. (3) `VPStem` carries the root form. The reason for separating these three cases is that they cannot occur at the same time. The label `'inf'` stores the required form of the verb in that corresponding tense, whereas `'fin'` stores the copula (auxiliary verb). The label `'obj'` on the other hand, stores the object of a verb and also the agreement information of the object. The label `'vType'` stores information about transitivity of a verb with `VType`, which include: intransitive, transitive or di-transitive:

```
param VType = VIntrans | VTrans | VDiTrans ;
```

The label `'comp'` stores the complement of a verb. Notice that it also inflects in number, gender and person (`Agr` is defined previously), whereas the label `'ad'` stores an adverb. Finally, `'embComp'` stores the embedded complement. It is used to deal with exceptions in the word order of Punjabi, when making a clause. For instance, if a sentence or a question sentence is a complement of a verb then it takes a different position in a clause; i.e. it comes at very end of the clause as shown in the example with bold-face:

oo$_{\text{she}}$ kehendi:$_{\text{say}}$ ae$_{\text{Aux}}$ **ke$_{\text{that}}$ mẽ$_{\text{I}}$ ro:ṭi$_{\text{bread}}$ k$^{\text{h}}$aṇḍa:$_{\text{eat}}$ wã$_{\text{Aux}}$**
She says that I (masculine) eat bread.

However, if an adverb is used as a complement of a verb then it comes before the main verb, as shown in the following example:

oo$_{\text{she}}$ kehendi$_{\text{say}}$ ae$_{\text{Aux}}$ ke$_{\text{that}}$ oo$_{\text{she}}$ **te:z$_{\text{briskly}}$** caldi:$_{\text{walks}}$ ae$_{\text{Aux}}$
She says that she walks briskly

Construction: The lexical category verb `(V)` is converted to the syntactic category verb phrase `(VP)` through different `(VP)` construction functions. The simplest is:

```
fun UseV : V →VP ; -- sleep, so:na:
lin UseV v = predV v ;
```

The function (predV) converts the lexical category (V) to the syntactic category (VP).

```
predV : Verb →VPH = \verb -> {
s = \\vh => case vh of {
  VPTense VPPres (Ag g n p) => fin =copula CPresent n p g;
     inf =verb.s!VF Imperf p n g ;
  VPTense VPPast (Ag g n p) => {
     fin = [] ; inf =verb.s!VF Perf p n g
    } ;
  VPTense VPFutr (Ag g n p) => {
     fin = copula CFuture n p g ;
     inf = verb.s ! VF Subj p n g
    } ;
  VPTense VPPerf (Ag g n p) => {
     fin = [] ; inf = verb.s!Root ++ cka g n
    } ;
  VPStem => { fin = [] ; inf = verb.s ! Root };
   _ => {fin = [] ; inf = verb.s!Root}
 };
 obj = {s = [] ; a = defaultAgr} ;
 vType = VIntrans ;
 ad = [] ;
 embComp = [] ;
 comp = \\_ => []
} ;
```

The lexical category (V) has three forms (corresponding to perfective/imperfective aspects and subjunctive mood). These forms are then used to make four forms (VPPres, VPPast, VPFutr, VPPerf in the above code) at the VP level, which are used to cover different combinations of tense, aspect and mood of Punjabi at the clause level. As an example, consider the explanation of the above code in bold-face. It builds a part of the inflection table represented by 's' for 'VPPres' and all possible combination of gender, number and person (Ag g n p). As shown above, the imperfective form of the lexical category (V) (i.e. VF Imperf p n g) is used to make the present tense at the (VP) level. The main verb is stored in the field labeled as 'inf' and the corresponding auxiliary verb (copula) is stored in the label 'fin'. All other parts of (VP) are initialized to default or empty values in the above code. These parts will be used to enrich the (VP) with other constituents, e.g.

adverb, complement etc. This is done in other (VP) construction functions including but not limited to:

```
Want to run, do:ṛna:_run ca:na:_want
ComplVV : VV →VP →VP;

Say that she runs, kena:_say ke_that oo_she do:ṛdi:_run ae_coupla
ComplVS : VS →S →VP; ,

Sleep here, ai:tʰe_here so:na:_sleep
AdvVP : VP →Adv →VP;
```

3.3.3 Adjectival Phrases

At morphological level, Punjabi adjectives inflect in number, gender and case [Humayoun and Ranta, 2010]. At syntax level, they agree with the noun they modify using the agreement information of a NP. An adjectival phrase (AP) can be constructed simply from the lexical category adjective (A) through the following function:

```
PositA : A →AP ; -- (Warm, garam)
```

Or from other categories such as:

```
Warmer than I, mi:re_I tõ_than garam_warm
ComparA : A →NP →AP ;

Warmer, garam
UseComparA : A →AP ;

As cool as Ali, ai:na:_as tʰanḍa:_cool jina:_as eli:_ali
CAdvAP : CAdv →AP →NP →AP ;
```

3.3.4 Adverbs and Closed Classes

The construction of Punjabi adverbs is very simple because "they are normally unmarked and don't inflect" [Humayoun and Ranta, 2010]. We have different construction functions for adverbs and other closed classes at both lexical and syntactical level. For instance, consider the following constructions:

```
Warmly, garam jo:xi:
fun PositAdvAdj : A →Adv ;
```

```
Very quickly, bahut_very ti:zi:_quickly de na:l_coupla
fun AdAdv : AdA →Adv →Adv ;
```

3.3.5 Clauses

While a phrase is a single word or group of words, which are grammatically linked to each other, a clause is a single phrase or group of phrases. Different types of phrases (e.g. `NP, VP`, etc.) are grouped together to make clauses . Clauses are then used to make sentences. In the GF resource grammar API tense system the difference between a clause and a sentence is: a clause has a variable tense, while a sentence has a fixed tense. We first construct clauses and then just fix their tense in order to make sentences. The most important function for the construction of a clause is:

```
PredVP : NP →VP →Cl ; -- Ali walks
```

The clause (Cl) has the following linearization type:

```
Clause : Type = {s : VPHTense => Polarity => Order =>Str} ;

Where:

param VPHTense = VPGenPres | VPImpPast | VPFut
     | VPContPres | VPContPast| VPContFut
     | VPPerfPres | VPPerfPast | VPPerfFut
     | VPPerfPresCont | VPPerfPastCon
     | VPPerfFutCont | VPSubj ;
param Polarity = Pos | Neg
param Order = ODir | OQuest
```

The tense system of GF resource library covers only eight combinations with four tenses (present, past, future and conditional) and two anteriorities (Anter and Simul). It does not cover the full tense system of Punjabi, which is structured around the aspect, tense, and mood. We make sentences in twelve different tenses (VPHTense in the above given code) at clause level to get a maximum coverage of the Punjabi tense system. Polarity is used to construct positive and negative, while 'Order' is used to construct direct and question clauses. We ensure the SOV agreement by saving all needed features in a (NP). These are made accessible in the PredVP function. A distinguishing feature of Punjabi SOV agreement is ergative behavior where transitive perfective verb may agree with the direct object instead of the sub-

ject. Ergativity is ensured by selecting the agreement features and noun-form accordingly. We demonstrate this in the following simplified code segment:

```
let subjagr : NPCase * Agr = case vt of {
  VPImpPast => case vp.subj of {
       VTrans   => <NPErg, vp.obj.a>;
       VDiTrans => <NPErg, defaultAgr> ;
       -        => <NPC Dir, np.a>
      }  ;
   -  => <NPC Dir, np.a>}
```

For perfective aspect `VPImpPast`, if a verb is transitive then it agrees with the object and therefore the ergative case of a NP is used (achieved through the line `VTrans => <NPErg, vp.obj.a>` in the above code). For DiTransitive verbs the agreement is set to the default but the ergative case is still needed (i.e. `VDiTrans =><NPErg, defaultAgr>`).

In all other cases (specified with the wild card "_" in the above code) the agreement is made with the subject (`np.a`), and we use the direct case (i.e. `NPC Dir`).

After selecting the appropriate forms of each constituent (according to the agreement features) they are grouped together to form a clause. For instance, consider the following simplified code segment combining different constituents of a Punjabi clause:

```
np.s!subj ++ vp.ad ++ vp.comp!np.a ++ vp.obj.s ++ nahim ++ vps.
   inf ++ vps.fin ++ vp.embComp;
```

Where: (1) `np.s!subj` is the subject; (2) `vp.ad` is the adverb (if any); (3) `vp.comp!np.a` is verb's complement; (4) `vp.obj.s` is the object (if any); (5) `nahim` is the negative clause constant; (6) `vps.inf` is the verb; (7) `vps.fin` is the auxiliary verb; (8) `vp.embComp` is the embedded complement.

3.4 Coverage and Limitations

The grammar we have developed consists of 44 categories and 190 syntax functions. It covers a fair enough part of the language but not the whole language. The reason for this limitation is the approach of a common abstract syntax defined for a set of languages in the GF resource library. Indeed it is not possible to have an abstract syntax, which is common to, and covers all features of the full set of languages. Consequently, the current grammar does not cover all aspects of Punjabi. However, this does not put any limitation on the extension of a language resource. It can be extended by implementing

language specific features in an extra language-specific module. These features will not be accessible through the common API, but can be accessed in Punjabi application grammars.

3.5 Evaluation and Future Work

It is important to note that completeness is not the success criteria for this kind of grammar based resource, but the accuracy is [Ranta, 2009a]. Evaluating a resource grammar is just like evaluating a software library in general. This type of evaluation is different from evaluation of a natural language processing application in general, where testing is normally done against some corpus. To evaluate the accuracy, we use the Punjabi resource grammar to translate, and observe, a test suite of examples from English to Punjabi. We achieved an accuracy of 98.1%. The reason for not having 100% accuracy is that our current grammar does not cover all aspects of the language. One such aspect is compound verbs of Punjabi, formed by nouns and the auxiliary verb 'to be' (hona:). In this case, the gender of auxiliary verb must agree with the inherent gender of the noun. We have not yet covered this agreement for compound verbs, which will produce incorrect translations. An interesting (yet wrong) example would be:

```
barix ho:nḍa: pe:a: ae (It is raining)
Instead of "ho:nḍa: pi:a:", it should be "honḍi: pai:"
```

Another, such feature is the repetitive use of verbs in Punjabi (e.g. munḍa:$_{\_boy}$ ru:nḍe$_{_weping}$ ru:nḍe$_{_weping}$ sũ$_{_slept}$ gi:a:$_{\_coupla}$, مُنڈا روندے روندے سوں گیا, the boy slept weeping). Coverage of such language specific details is one direction for the future work.

3.6 Related Work and Conclusion

In general language resources for Punjabi are very limited; especially for the one spoken in Pakistan and written in Shahmukhi. Furthermore, most of the applications related to Punjabi are designed only for the Punjabi, written and spoken in India; hence, only support the Gurmukhi script. A review of such applications is given in [Lehal., 2009]. There are some attempts to interchange between these scripts with transliteration systems. However, the current systems only seem to provide partial solutions, mainly because of the vocabulary differences [Humayoun and Ranta, 2010]. A transfer-based machine translation system reported in [Lehal., 2009] translates between Punjabi and Hindi only. On the contrary, the Punjabi resource grammar is based

on an Inter-lingua approach, which makes it possible to translate between seventeen languages in parallel. With the best of our knowledge, this work is the first attempt to implement a computational Punjabi grammar as open source.

We have described implementation of the computational grammar for Punjabi. Punjabi is an under-resourced language. So, this work might be a useful resource, and may encourage other researchers to work in this direction. As the resource grammar does not cover full features of Punjabi, it is not possible to use it for parsing and translation of arbitrary text. However, it is best suited for building domain specific application grammars.

Chapter 4

An Open Source Persian Computational Grammar

This chapter is based on a conference paper and describes the development of the Persian resource grammar. Other than describing the development of different general-purpose grammatical constructions of Persian, it also describes how to deal with distinctive features, such as *ezafe* construction, in Persian.

The layout has been revised.

Abstract: In this paper, we describe a multilingual open-source computational grammar of Persian, developed in the Grammatical Framework (GF) – A type-theoretical grammar formalism. We discuss in detail the structure of different syntactic (i.e. noun phrase, verb phrase, adjectival phrase, etc.) categories of Persian. First, we show how to structure and construct these categories in GF. Then, we describe how they are glued together to make well-formed sentences in Persian, while maintaining the grammatical features such as agreement, word order, etc. We also show how some of the distinctive features of Persian, such as the *ezafe* construction, are implemented in GF. In order to evaluate the grammar's correctness, and to demonstrate its usefulness, we have added support for Persian in a multilingual application grammar (Phrasebook) using the reported resource grammar.
Keywords: Grammatical Framework, Abstract syntax, Concrete syntax.

4.1 Introduction

The idea of providing assistance to programmers in the form of software libraries is not new. It can be traced back to 1959, when JOVIAL[1] gave the concept of COMPOOL (Communication Pool). In this approach, the code and data that provide independent services are made available in the form of software libraries. Software libraries are now at the heart of modern software engineering, and many programming languages (e.g. C, C++, Java, Haskell, etc.) come with built-in libraries. However, the idea of providing natural language grammars as software libraries is new in GF (Grammatical Framework) [Ranta, 2004].

GF is a special purpose programming language designed for developing natural language processing applications. Historically, GF and its libraries have been used to write a number of application grammars including GF-Key – a tool for authoring and translation of software specifications, TALK – a multilingual and multimodal spoken dialogue system, and WebALT – an application grammar for multilingual generation of mathematical exercises. Moreover, GF has support for an increasing number of natural languages. Currently, it supports 23 languages (see the status[2] of GF resource library for more details). GF provides libraries in the form of `resource grammars` – one of the two types of programs that can be written in GF. A resource grammar is a general-purpose grammar [Ranta, 2009b] that encodes the syntactic constructions of a natural language. For example modification of a noun by

[1]JOVIAL is a high-order programming language specialized for the development of embedded systems

[2]http://www.grammaticalframework.org/lib/doc/status.html

an adjective is a syntactic construction, and it is developed as a part of a resource grammar development. A collection of such syntactic constructions is called a resource grammar. A resource grammar is supposed to be written by linguists, who have sufficient grammatical knowledge (i.e. knowledge about word order, agreement features, etc.) of the target natural language. The other type of grammar that one can write in GF is an `application grammar`. It is a domain specific grammar that encodes semantic constructions. It is supposed to be written by domain experts, who have a better understanding of the domain specific terms. However, an application grammar may use a resource grammar as a supporting library [Ranta, 2009a] through a common resource grammar API .

Furthermore, every grammar in GF has two levels: `abstract syntax` and `concrete syntax`, which are based on Haskell Curry's distinction of `tectogrammatical` and `phenogrammatical` structures [Curry, 1961]. The abstract syntax is independent of any language and contains a list of categories (cat), and a set of tree-defining functions (fun). The concrete syntax contains rules telling how the abstract syntax categories and trees are linearized in a particular language. Since the abstract syntax is common to a set of languages – languages that are part of the GF resource library – it becomes possible to have multiple parallel concrete syntaxes for one abstract syntax. This makes the GF resource library multilingual. Development of a resource grammar means writing linearization rules (i.e. lincat and lin) of the abstract syntax categories and functions (i.e. cat and fun), for a given natural language. This is a challenging task, as it requires comprehensive knowledge of the target natural language as well as practical programming experience of GF. In this paper, we describe the development of the Persian resource grammar.

Persian is an Iranian language within the Indo-Iranian branch of the Indo-European family of languages. It is widely spoken in Iran, Afghanistan, Tajikistan, and Uzbekistan. In Iran it is also called Farsi, and the total number of Farsi speakers is about 60 million [Bahrani et al., 2011]. It has a suffix predominant morphology, though there are a small number of prefixes as well [Megerdoomian, 2000]. Persian tense system is structured around tense, aspect and mood. Verbs agree with their subject in number and person, and there is no grammatical gender [Mahootiyan, 1997]. Persian has a relatively free word order [Müller and Ghayoomi, 2010], but declarative sentences are mostly structured as "(S) (O) (PP) V". Optional subject (S) is followed by an optional object (O), which is followed by an optional propositional phrase (PP). All these optional components precede the head verb (V).

The paper is structured as follows: In Sections 4.2 and 4.3, we talk about morphology and syntax (two necessary components of a grammar) followed

by an example in Section 4.4. Coverage and evaluation is discussed in Section 4.5, while related and future work follows in Section 4.6.

4.2 Morphology

Every GF resource grammar has a test lexicon of almost 450 words. These words belong to different lexical categories (both open and closed), and have been randomly selected for test purposes. Different inflectional forms of these words are built through special functions called lexical paradigms. These lexical paradigms take the canonical form of a word and build finite inflection tables. However, the morphological details are beyond the scope of this paper and it concentrates on the syntactical details.

4.3 Syntax

While morphology is about principles and rules of making individual words, syntax is about how these words are grouped together to make well-formed sentences in a particular language. In this section, we describe the syntax of Persian. First, in the following subsections we discuss different syntactic categories (i.e. noun phrases, verb phrases, adjectival phrases, etc.) individually. Then, we show how they are glued together to make clauses and sentences in sections 4.3.5 and 4.3.6 respectively.

4.3.1 Noun Phrase

A noun phrase is a single word or a group of grammatically related words that function as a noun. It consists of a head noun, which is constructed at the morphological level, and one or more optional modifiers. In Persian, modifiers mostly follow the noun they modify, even though in limited cases they can precede it. Below, we show the structure of a noun phrase (NP) in our implementation, followed by its construction.

Structure: A noun phrase (NP) has the following structure:

```
cat NP ;
lincat NP:Type = {s  : NPForm=>Str;
                  a  : AgrPes ;
                  animacy : Animacy };
Where
param NPForm = NPC Ezafe ;
```

```
param Ezafe = bEzafe | aEzafe | enClic;
param AgrPes = AgPes Number PPerson;
param Number = Sg | Pl;
param PPerson = PPers1 | PPers2| PPers3;
param Animacy = Animate | Inanimate ;
```

The curly braces show that a NP is a record of three fields. The purpose of different fields of a NP is explained below:

- `'s'` defined as `s:NPForm=>Str` is interpreted as: `s` is an object of the type `NPForm=>Str`, where the type `NPForm=>Str` is a table type structure. In GF, we use such table type structures to formulate inflection tables. In brief `s` stores different forms of a noun phrase corresponding to the parameters `bEzafe` (a form without the ezafe suffix), `aEzafe` (a form with the ezafe suffix) and `enClic` (a form with the enclitic particle). For example consider the following table for the noun `house`.

  ```
  s . NPC bEzafe =>خانه -- χaːnæh
  s . NPC aEzafe =>خانه ی  -- χaːnæhiː
  s . NPC enClic =>خانه ای -- χaːnæhaːiː
  a . AgPes Sg PPers3
  animacy . Animate
  ```

  These forms are then used in the construction of clauses and/or other categories. For example, in Persian the `aEzafe` form is used in modifications like, adjectival modification (e.g. خانه ی بزرگ χaːnæhiː bazorg `, big house`), and in showing possession (e.g. خانه ی من, χaːnæhiː man`, my house`). The `enClic` form is used in constructions where the noun is followed by a relative clause e.g. خانه ای که آنجا است, χaːnæhaːiː keh aːnjaː aːsat`, the house which is there`.

- `'a'` is the agreement parameter and stores information about number and person of a noun phrase. This information is used for agreement with other categories.

- `'animacy'` keeps the information about whether the head noun in the noun phrase is animate or inanimate. This information is useful in the subject-verb agreement at the clause level.

Construction: The head noun corresponds to the morphological category noun `N`. The morphological category `N` is first converted to an intermediate category called common noun `CN`, through the following function:

```
fun UseN : N -> CN ; -- خانه ,χaːnæh, house
```

Where a common noun has the following structure:

```
lincat CN = {s : Ezafe=>Number=>Str ; animacy : Animacy};
```

It deals with modification of a noun by different modifiers including but not limited to adjectives, quantifiers, determiners, etc. We have different functions for these modifications. Consider the following function that is used for adjectival modification:

```
fun AdjCN : AP -> CN -> CN ;
-- خانه ی بزرگ χaːnæhiː bzorg , big house
```

And its linearization rule for Persian is given below:

```
lin AdjCN ap cn = {
 s = table { bEzafe => table {
      Sg => cn.s ! aEzafe ! Sg ++ ap.s ! bEzafe;
      Pl => cn.s ! aEzafe ! Pl ++ ap.s ! bEzafe
                                        };
               aEzafe => table {
      Sg => cn.s ! aEzafe ! Sg ++ ap.s ! aEzafe;
      Pl => cn.s ! aEzafe ! Pl ++ ap.s ! aEzafe
                                      };
                  enClic => table {
      Sg => cn.s ! aEzafe ! Sg ++ap.s ! enClic;
      Pl => cn.s ! aEzafe ! Pl ++ ap.s ! enClic
                                         };
             };
animacy = cn.animacy
                        };
```

The above linearization rule takes an adjectival phrase and a common noun and builds a modified common noun. Again, `'s'` in the above given code is an inflection table from `Ezafe` to `Number` to `String`, and stores different inflectional forms of a modified common noun. Since Persian adjectives do not inflect for number, we use the same form of an adjective, both for `Sg` and `Pl` parameters of the common noun. However, adjectives have three forms corresponding to `bEzafe`, `aEzafe` and `enClic` (see Section 4.3.3). As it is clear in the above code, whenever a common noun is modified by an adjective, the `aEzafe` form of the common noun is used (i.e. cn.s ! aEzafe ! Sg). Moreover, the modifier follows the common noun to ensure the proper word order. GF provides a syntactic sugar for writing the above table concisely. For example the above given code can be replaced by the following simplified version.

```
lin AdjCN ap cn = {
    s = \\ez,n => cn.s ! aEzafe ! n ++ ap.s ! ez;
    animacy = cn.animacy
     } ;
```

Note how the \\ operator is used as a syntactic sugar with parameter variables `ez` and `n` to compress the branches of a table together. Also note that the symbol `!` is used as a selection operator to select different values from the inflection table and `++` is used as a concatenation operator.

The resulting common noun is then converted to a noun phrase `(NP)` through different functions depending on the constituents of a NP. In the simplest case a common noun without any article can be used as a mass noun phrase. It is constructed through the following function:

```
fun MassNP : CN -> NP ; -- آب , aːb, water
```

And its linearization rule is:

```
lin MassNP cn = {s = \\ez => cn.s ! ez ! Sg
               a   = AgPes PPers3 Sg ;
               animacy = cn.animacy
                         } ;
```

This function takes a common noun and converts it to a NP.
Few others functions for the construction of a NP are:

```
fun DetCN : Det -> CN -> NP ;
   -- مرد mard, man
fun AdvNP : NP -> Adv -> NP ;
   -- پاریس امروز pariːs amroːz, Paris today
fun DetNP : Det -> NP ;
    -- این پنج  ,aːn panj, these five
```

4.3.2 Verb Phrase

A verb phrase normally consists of a verb and one or more optional complements. It is the most complicated category in our constructions. First, we explain the structure of the Persian verb phrase in detail, and then we continue with the description of its construction.
Structure: In our construction a verb phrase `VP` has the following structure:

```
cat VP ;
lincat VP : Type = {
  s   : VPHForm => {inf : Str} ;
  obj  : Str ;
```

```
  comp  : AgrPes => Str;
  vComp : AgrPes => Str;
  embComp : Str ;
  inf   : Str;
  adv   : Str;
  } ;

Where

param VPHForm = VPTense Polarity VPPTense AgrPes
                           | VPImp Polarity Number
                           | VVForm AgrPes
                          | VPStem1
                          | VPStem2 ;
param VPPTense = VPPres Anteriority
     |VPPast Anteriority
     |VPFutr Anteriority
     |VPCond Anteriority ;
param Anteriority = Simul | Anter ;
```

A brief explanation of different fields and their purpose is given below:

- Once again, `'s'` is an inflection table, and here, it stores the actual verb form. We make different forms of a verb at the verb phrase level. The parameter `VPHForm` in the above code stores these different forms. A brief overview of these forms and their usage is given below:
  - `'VPTense'` is a constructor with context parameters `Polarity`, `VPPTense` and `AgrPes`. It stores different forms of a verb inflecting for `polarity`, `tense` and `AgrPes` (where `AgrPes = AgPes Number PPerson`). These forms are used to make nominal declarative sentences at the clause level.
  - `'VPImp'` stores the imperative form of a verb inflecting for polarity and number.
  - `'VVForm'` stores the form of a verb, which is used when a verb takes the role of a complement of another verb (i.e. in the construction "want to run", 'to run' is used as a complement of the auxiliary verb 'want'. In English the infinitive of the second verb ('to run') is used as the complement of the auxiliary verb ('want'), but in Persian in most cases the present subjunctive form of the second verb is used as the complement of the auxiliary verb. We name this form as the `VVForm`. It inflects for number and person.

  - Finally `'VPStem1'` and `'VPStem2'` store the present and past roots of a verb, which have different forms in Persian.

- `'obj'` is a string type field, which stores the direct object of a verb.

- `'comp'` is an inflection table which stores the complements of a verb; those other than a direct object. The complement needs to be in agreement with the subject both in number and person. Therefore, we keep all the inflectional forms (inflecting for number and person) of a complement. This parameter is used to store indirect objects of di-transitive verbs.

- `'vComp'` is another inflection table inflecting for number and person. When a verb is used as a complement of an auxiliary verb, we store it in this field. Unlike `comp` or `obj`, this type of complement follows the auxiliary verb. For example in the sentence او می خواهد بخوابد, a:u: mi: xu:a:had bxu:a:bad, `she wants to sleep`, the verb خوابیدن, xu:a:bi:dan `, to sleep` is the complement of the auxiliary verb خواستن , xva:stan`, want`, therefore it will follow the auxiliary verb.

- `'embComp'` is a simple string and is used when a declarative or interrogative sentence is used as a complement of a verb. For example in the sentence او می گوید که من دارم می خوابم, a:u: mi: goi:d keh man da:ram mi: xu:a:ham`, she says that I am sleeping`, the sentence من دارم می خوابم , man da:ram mi: xu:a:ham `, I am sleeping` is a complement of the verb گفتن goftan`, to say`. This type of complement comes at the very end of a clause. The reason behind storing different types of complements in different fields is that in Persian these different complements take different positions within a clause (see section 4.3.5 for more details).

- `'inf'` simply stores the infinitive form of a verb.

- `'adv'` is a string field and stores an adverb.

Construction: The verb phrase `VP` is constructed from the morphological category verb `V` by providing its complements. In the simplest case, a single verb without any complements can be used as a verb phrase. We create this verb phrase through the following function:

```
fun UseV : V -> VP ;
  خوابیدن  xu:a:bi:dan, sleep
```

And its linearization rule is:

```
lin UseV v = predV v ;

Where

oper predV : Verb -> VP = \verb -> {
 s = \\vh =>
    case vh of {
      VPTense pol (VPPres Simul) (AgPes n p) => inf = verb.s !
          VF pol (PPresent PrImperf) p n ;
      VPTense pol (VPPres Anter) (AgPes n p) =>
{ inf = verb.s ! VF pol (PPresent PrPerf) p n } ;
      VPTense pol (VPPast Simul) (AgPes n p) =>
{ inf =verb.s ! VF pol (PPast PstAorist) p n } ;
        VPTense pol (VPPast Anter) (AgPes n p) =>
{ inf =verb.s ! VF pol (PPast PstPerf) p n } ;
      VPTense pol (VPFutr Simul) (AgPes n p) =>
{ inf =verb.s ! VF pol (PFut FtAorist) p n } ;
     VPTense pol (VPFutr Anter) (AgPes n p) =>
{ inf = verb.s ! VF pol (PPresent PrPerf) p n } ;
     VPTense pol (VPCond Simul) (AgPes n p) =>
{ inf = verb.s ! VF pol (PPast PstImperf) p n } ;
     VPTense pol (VPCond Anter) (AgPes n p) =>
{ inf = verb.s ! VF pol (PPast PstImperf) p n } ;
     VPImp pol n => { inf = verb.s ! Imp pol n} ;
     VVForm (AgPes n p) =>
{inf = verb.s ! Vvform (AgPes n p)} ;
     VPStem1 => { inf = verb.s ! Root1};
     VPStem2 => { inf = verb.s ! Root2}
                };
 obj = {s = [] ; a = defaultAgrPes} ;
 comp = \\_ => [];
 vComp = \\_ => [] ;
 embComp = [];
 inf = verb.s ! Inf;
 adv = [];
} ;
```

This operation (indicated by keyword `oper` in the above code) converts a verb (a morphological category) to a verb phrase (a syntactic category). At the morphological level, Persian verbs inflect for tense `present/past/future`, aspect `perfective/imperfective/aorist`, polarity `positive/negative`, per-

son `1st/2nd/3rd`, and number `Sg/Pl`. All these morphological forms are stored in an inflection table at the morphological level, and are used in this operation to make different forms at the verb phrase level. For example, the boldfaced line in the above code builds a part of the inflection table `s`. This part stores the forms of the verb that correspond to the (Present, Simul) combination of tense and anteriority, and all possible combinations of polarity and agreement (represented by variables 'pol' for polarity and `AgPes n p` for agreement).

All the complement fields of this verb phrase are left blank or initialized to default values. These complements are provided through other verb phrase construction functions including but not limited to the followings:

```
fun ComplVV : VV -> VP -> VP ;
  او می خواهد بدود  (a:u:) mi: xu:a:had badu:d, want to run
fun ComplVS : VS -> S -> VP ;
  او می گوید او می دود  (a:u:) mi: gu:i:d a:u: mi: du:d, say that she runs
fun ComplVQ : VQ -> QS -> VP ;
  او در تعجب است چه کسی می دود ,(a:u:) dr tjab a:st ceh kisi:
     mi:dod , wonder who runs
```

These functions enrich the verb phrase by providing complements. The resulting verb phrase is then used in making clauses, which is discussed in section 4.3.5.

4.3.3 Adjectival Phrase

In our construction an adjectival phrase has the following structure:

```
lincat AP = {s : Ezafe => Str ; adv : Str} ;
```

Again `s` stores different forms corresponding to the parameters: `bEzafe` (before Ezafe), `aEzafe` (after Ezafe), and `enClic` (enclitic). `adv` is a string field which stores the corresponding form, which is used when an adjective is used as an adverb. Adjectival phrases are constructed from the morphological category adjective (A) through different construction functions. The simplest one is:

```
fun PositA : A -> AP ; -- گرم , garam , warm
```

This function simply converts the morphological category adjective `A` to the syntactic category adjectival phrase `AP`. Its linearization rule for Persian is very simple because an adjective and an adjectival phrase have the same structure. This is as simple as given below:

```
lin PositA a = a ;
```

Further, it is possible to construct adjectival phrases from other categories. We have one function for each corresponding construction including the followings:

```
fun ComparA : A -> NP -> AP ;
    گرم تر از من  garam tar aːz man , warmer than I
fun AdjOrd : Ord -> AP ;
  گرم ترین  garam triːn, warmest
fun CAdvAP : CAdv -> AP -> NP -> AP ;
     به جالبی جان  beh jaːlbiː jaːn, as cool as John
fun AdAP : AdA -> AP -> AP ;
    □ خیلی گرم χiːliː garam, very warm
```

4.3.4 Adverbs and other Closed Categories

Adverbs are made at morphological level, but it is also possible to construct them at syntactic level form other categories, for example from adjectives. We have separate construction functions for adverbs and other closed categories e.g. pronouns, quantifiers, etc. A few of them are listed here:

```
fun PositAdvAdj : A -> Adv ;
  به گرمی  beh garamiː , warmly
fun PossPron : Pron -> Quant ;
خانه ی من  , (χaːnæh iː) man, my (house)
fun AdvIP : IP -> Adv -> IP ;
چه کسی در پاریس  ceh kisiː dar paːriːs, who in Paris
```

4.3.5 Clauses

While a phrase is a single word or a group of grammatically related words, a clause is a single phrase or a group of phrases. Another difference is that a clause may have both a subject and a predicate of its own, while a phrase cannot have both at the same time. Though, sometimes it is possible that a clause does not have any subject at all, and is only composed of a verb phrase.

Structure: In our construction a clause has the following structure:

```
lincat Clause : Type = {s : VPTense => Polarity =>
Order => Str} ;

Where
```

```
Param VPHTense = VPres |VPas |VFut |VPerfPres
               |VPerfPast |VPerfFut|VCondSimul
               |VCondAnter ;
```

`'s'` stores clauses with variable tense, polarity and order (`declarative/interrogative`), which are fixed at the sentence level. The GF resource grammar API tense system covers only 8 possibilities through the combination of four tenses (present, past, future and conditional) and two anteriorities (anter and simul). The common API tense system is not adequate for Persian tense system - which is structured around tense, aspect, and mood. However, in our current implementation we stick to the common API tense system, and thus cover only eight possibilities. A better approach is to implement the full tense system of Persian and then map it to the common resource API tense system. This approach has been applied in the implementation of Urdu [Shafqat et al., 2010] and Punjabi [Shafqat et al., 2011] tense systems.

Construction: A clause is constructed through different clause construction functions depending on the constituents of a clause. The most important construction is from a noun phrase `NP` and a verb phrase `VP` through the following function:

```
fun PredVP : NP -> VP -> Cl ;
```
جان راه می رود jaːn raːh miːruːd, John walks

And its linearization rule for Persian is:

```
lin PredVP np vp = mkClause np vp ;

Where

oper mkClause : NP -> VP -> Clause = \np,vp -> {
  s = \\vt,pol,ord =>
  let
  subj = np.s ! NPC bEzafe;
  agr = np.a ;
  vps = case <pol,vt> of {
<Pos,VPres> =>
vp.s ! VPTense Pos (VPPres Simul) agr ;
<Neg,VPres> =>
vp.s ! VPTense Neg (VPPres Simul) agr ;
<Pos,VPerfPres>=>
vp.s ! VPTense Pos (VPPres Anter) agr;
<Neg,VPerfPres> =>
```

```
vp.s ! VPTense Neg (VPPres Anter) agr;
<Pos,VPast> =>
vp.s ! VPTense Pos (VPPast Simul) agr ;
<Neg,VPast> =>
vp.s ! VPTense Neg (VPPast Simul) agr ;
<Pos,VPerfPast>=>
vp.s ! VPTense Pos (VPPast Anter) agr;
<Neg,VPerfPast>=>
vp.s !VPTense Neg (VPPast Anter) agr;
<Pos,VFut> => case vp.wish of {
  True => vp.s ! VPTense Pos (VPPres Simul) agr ;
  False => vp.s ! VPTense Pos (VPFutr Simul) agr};
<Neg,VFut> => case vp.wish of {
True => vp.s ! VPTense Neg (VPPres Simul) agr;
False => vp.s ! VPTense Neg (VPFutrSimul) agr};
<Pos,VPerfFut> => case vp.wish of {
True => vp.s ! VPTense Pos (VPPres Anter) agr ;
False => vp.s ! VPTense Pos (VPFutr Anter) agr};
<Neg,VPerfFut> => case vp.wish of {
True => vp.s ! VPTense Neg (VPPres Anter) agr ;
False => vp.s ! VPTense Neg (VPFutr Anter) agr};
<Pos,VCondSimul> =>
vp.s ! VPTense Pos (VPCond Simul) agr;
<Neg,VCondSimul> =>
vp.s ! VPTense Neg (VPCond Simul) agr;
<Pos,VCondAnter> =>
vp.s ! VPTense Pos (VPCond Anter) agr;
<Neg,VCondAnter> =>
vp.s ! VPTense Neg (VPCond Anter) agr };
quest = case ord of
            { ODir => []; OQuest => "أيا" };
in
quest ++ subj ++ vp.adv ++ vp.comp ! np.a ++ vp.obj.s ++
   vps.inf ++ vp.vComp ! np.a ++ vp.embComp
};
```

This operation takes a noun phrase `NP` and a verb phrase `VP` and constructs a clause with variable tense, polarity and order. Note how agreement information of the noun phrase (i.e. `np.a` in the above code) is used to select the appropriate form of the verb phrase. This is done to ensure the subject-verb agreement. The `let` statement stores different constituents of a verb phrase

in different variables. Once we have all these constituents, they can be combined with the subject noun phrase in order to make a clause (see boldfaced code segment). Also note that in the declarative clauses the `bEzafe` (before Ezafe) form of the subject noun phrase (i.e. `subj` in the above code) is used. As an example, if the noun phrase (John) and the verb phrase (walk) were inputs to the above function, the output would be the following clause (only a portion of the full clause is shown):

```
s . VPres => Pos => ODir => جان راه می رود
  -- ja:n ra:h mi: ru:d , John walks
s . VPres => Pos => OQuest =>  آیا جان راه می رود
  -- a:i:a: ja:n ra:h mi: ru:d, Does John walk?
s . VPres => Neg => ODir => جان راه نمی رود
  -- ja:n ra:h nami: ru:d, John does not walk.
s . VPres => Neg => OQuest => آیا جان راه نمی رود
   -- a:i:a: ja:n ra:h nami: ru:d, Does John not walk?
s . VPast => Pos => ODir => جان راه رفت
  -- ja:n ra:h raft , John walked.
s . VPast => Pos => OQuest => آیاجان راه رفت
  -- a:i:a: ja:n ,ra:h raft, Did John walk?
s . VPast => Neg => ODir => جان راه نرفت
  -- ja:n ra:h nraft, John did not walk.
s . VPast => Neg => OQuest => آیا جان راه نرفت
   -- a:i:a: ja:n ra:h nraft, Did John not walk?
s . VFut => Pos => ODir => جان راه خواهد رفت
  -- ja:n ra:h xu:a:hd raft , John will walk.
s . VFut => Pos => OQuest =  آیا جان راه خواهد رفت
  -- a:i:a: ja:n ra:h xu:a:hd raft , Will John walk?
s . VFut => Neg => ODir => جان راه نخواهد رفت
  -- ja:n ra:h nxu:a:hd raft , John will walk.
s . VFut => Neg => OQuest =>  آیا جان راه نخواهد رفت
  -- a:i:a: ja:n ra:h nxu:a:hd raft ,Will John not walk?
s . VPerfPres => Pos => ODir => جان راه رفت است
  -- ja:n ra:h raft a:st , John has walked.
s . VPerfPres => Pos => OQuest => آیا جان راه رفت است
  -- a:i:a: ja:n ra:h raft a:st , Has John walked?
s . VPerfPres => Neg => ODir => جان راه نرفت است
  -- ja:n ra:h nrft a:st , John has not walked.
s . VPerfPres => Neg => OQuest =>  آیا جان راه نرفت است
  -- a:i:a: ja:n ra:h nraft a:st , Has John walked?
s . VPerfPast => Pos => ODir =>بود جان راه رفت
```

```
  -- jaːn raːh raft buːd , John had walked.
s . VPerfPast => Pos => OQuest => آیا جان راه رفت بود
  -- aːiːaː jaːn raːh raft buːd , Had John walked?
s . VPerfPast => Neg => ODir => بود جان راه نرفت
  -- jaːn raːh nraft buːd , John had not walked.
s . VPerfPast => Neg => OQuest => جان راه نرفت بود
  -- aːiːaː jaːn raːh nraft buːd , Had John not walked?
s . VPerfFut => Pos => ODir => جان راه رفت است
  -- jaːn raːh raft aːst , John will has walked.
s . VPerfFut => Pos => OQuest => آیا جان راه رفت است
  -- aːiːaː jaːn raːh raft aːst ,Will John has walked?
```

This covers only one way of making clauses, there exist others as well, for example:

```
fun PredSCVP : SC -> VP -> Cl ;
```

این که او می رود خوب است aːiːn kh aoː miː ruːd xuːb aːst,it is good that she goes.

4.3.6 Sentences

As mentioned and shown previously, a clause has variable tense, polarity, and order. Fixing these parameters results in declarative sentences. This is done through different functions, where the most important one is as follows:

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

The parameter `Temp` is a combination of two parameters: one for tense and the other for anteriority. Thus, the function `UseCl` takes tense, anteriority, polarity and a clause as its input and produces a sentence as output. Therefore, if we fix the variable features of the example clause given in the `Clause` section, we will get the following sentence - where tense is fixed to simple present, anteriority to simul, and polarity to positive.

```
s . جان راه می رود , jaːn raːh miː ruːd , John walks
```

This shows how we can make declarative sentences. Other types of sentences, i.e. interrogative sentences and relative sentences are built through the following functions respectively:

```
UseQCl : Temp -> Pol -> QCl -> QS ;
UseRCl : Temp -> Pol -> RCl -> RS ;
```

4.4 An Example

We give an example to demonstrate how our Persian resource grammar works at morphology and syntax levels. Consider the translation of the following sentence from English to Persian.
"He lives in my house"
Figure 4.1 shows the automatically generated parse tree of the above sentence. At the lowest level we have the lexical entries. These lexical entries

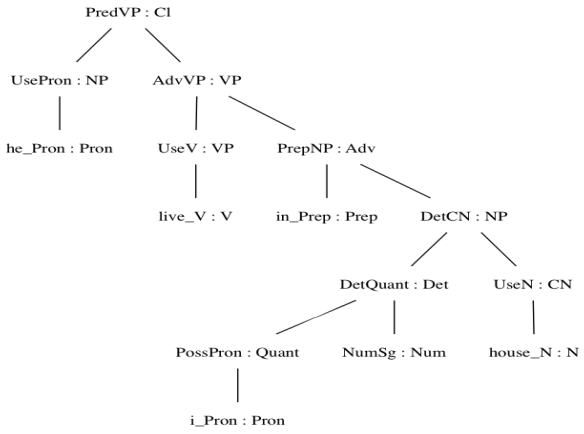

Figure 4.1: Parse Tree

are used to construct different syntactic categories. These constructions are made according to the grammatical rules, which are declared at the abstract-level. For example the category noun phrase `NP` can be built from a `Det` (determiner) and a `CN` (common noun). In the abstract syntax we have the following rule for this construction:

```
fun DetCN : Det -> CN -> NP ;
```

Our goal, as a resource grammar developer, is to provide the correct linearization rule for this abstract tree-building function in Persian. This is achieved through implementation of the concrete syntax (described in the syntax section) for Persian. The morphological part ensures that the correct forms of words are created, while the syntactical part handles other grammatical features such as agreement, word order, etc. Figure 4.2 shows an automatically generated word alignments for the example sentence: “he lives in my house”. The language pair is (English, Persian).

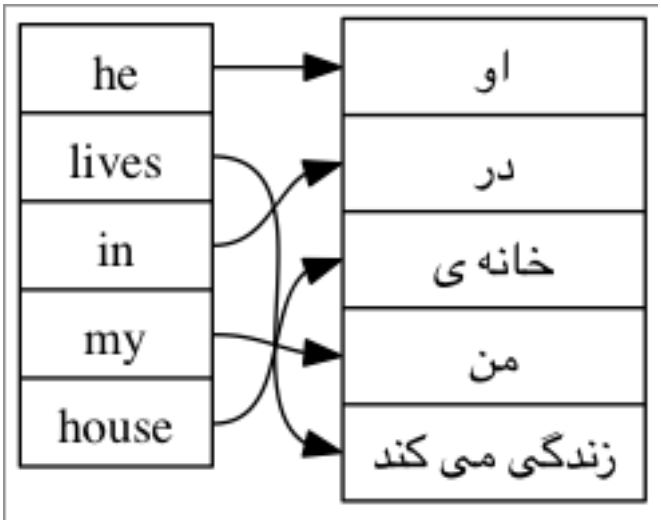

Figure 4.2: Word Alignments

4.5 Coverage and Evaluation

Our Persian resource grammar has 44 different categories and 190 syntax functions to cover different syntactic constructions. This covers a fair enough part of a language but not everything. The reason for not being able to cover the whole language is the chosen approach of a common abstract syntax for a set of languages in the resource grammar library. In principle, this approach makes it impossible to cover every aspect of every language. An example missing construction for Persian is the causative construction. Such missing constructions are supposed to be implemented in an extra language specific module, which is one direction for future work.

Testing a resource grammar is different from testing NLP applications in general, where testing is done against some text corpus. Testing resource grammars is much like testing software libraries [Ranta, 2009a]. In this type of testing, a library is tested by developing some application grammars on top of the resource grammars. Phrasebook [Ranta et al., 2012] is a multilingual application grammar that was developed as part of the MOLTO-Project. This application grammar has support for 15 languages. In order to evaluate our resource grammar we have added support for Persian to it. We achieved satisfactory results when a test case of 250 examples was generated. The application is open to test the accuracy and quality of translations, and is available on the MOLTO homepage[3]. Another possible way of testing is to

[3]http://www.molto-project.eu/

generate a set of trees, linearize them, and observe their correctness. This approach has been applied to produce examples for the synopsis[4] document, which contains a set of translated examples. The grammar was released when we reached a satisfactory performance level, with some known issues reported in the library documentation.

4.6 Related and Future Work

A Persian computational grammar was reported in [Bahrani et al., 2011]. This grammar is based on Generalized Phrase Structure Grammar (GPSG) model. Considering nouns, verbs, adjectives, etc. as basic structures, X-bar theory is used to define noun phrases, verb phrases, adjectival phrases, etc. This grammar is monolingual and can be used in applications, which need a syntactic analysis of the language. On the contrary, the grammar we developed is multilingual and can be used to develop different kinds of application grammars, ranging from text-translators to language generation applications, dialogue systems, etc.

[Müller and Ghayoomi, 2010] reported a Persian grammar implemented in TRALE system [Meurers et al., 2002]. The grammar is based on the Head-driven Phrase Structure Grammar (HPSG) and is still under construction. Its coverage is limited due to the missing lexical items (i.e. verbs, numerals, clitic forms of a copula, etc.)

As mentioned previously, the reported grammar does not cover all aspects of Persian. One direction for future work is to explore missing constructions and implement them in a separate language specific module. Another possible direction for future work is the development of more application grammars on top of the reported resource grammar.

[4] http://www.grammaticalframework.org/lib/doc/synopsis.html

Chapter 5

Lexical Resources

This chapter describes the development of different types of lexicons in GF. We use data from existing lexical resources (e.g. Dictionaries and WordNets) to produce uni-sense and multi-sense morphological lexicons. The size of these lexicons ranges from 10 to 50 thousand lemmas.

5.1 Introduction

The quality and coverage of a natural language processing (NLP) application depend hugely on the quality and coverage of its lexical resources. Words[1], as lexical units, play the role of basic building blocks. These building blocks should be in a proper shape and order if one is to achieve high quality in NLP applications. In the recent past, there have been many attempts to build reusable high quality lexical resources. Examples include the Princeton WordNet [Miller, 1995], the EuroWordNet [EuroWordNet, 1996], the Indo-WordNet [Bhattacharyya, 2010], and the VerbNet [Schuler, 2005] (see the Global Word-Net association[2] for a full list of such resources). Even though these resources provide many valuable semantic and lexical information about words and their relationships, what they do not provide is full-form morphological lexicons. They might have separate morphological processing functions to analyze the words, but these functions are normally not intended to be used for word-forms generation.

The GF resource grammar library API provides special functions called smart paradigms [Détrez and Ranta, 2012], which can be used to build full-form mono-lingual or multi-lingual GF lexicons. These paradigms take one or more forms of a word, try to analyze it, and build full-form inflection tables using different language-dependent word-form generation rules. In this chapter, we report two types of GF lexicons: *Mono-Lingual* and *Multi-Lingual*. In the next section, first we describe the structure and development of a GF lexicon in general, which is followed by the development of a mono-lingual and a number of multi-lingual lexicons.

5.2 GF Lexicons

As mentioned in the previous chapters, every GF grammar has two levels of representation: *abstract* and *concrete*. This applies to both the syntactical and the lexical modules. In the *abstract* representation of a lexical module, the words are declared as constants of a particular lexical category (i.e. noun, verb, adjective etc.). This representation is independent of any language, however for the purpose of convenience, the constant names are chosen to be in English. As an example, consider the following small segment from an abstract representation of a GF lexicon.

[1]The notion of a 'word' in itself is fuzzy, here we take it as a smallest unit that we process at the lexical level.

[2]http://www.globalwordnet.org/gwa/wordnet_table.html

Infinitive	Present 3Sg	Past	Past Participle	Present Participle
learn	learns	learned	learned	learning

Table 5.1: Inflectional forms of the verb 'learn'

```
abstract DictAbs = {

  fun boy_N : N ;
  fun lesson_N : N ;
  fun tall_A : A ;
  fun learn_V2 : V2 ;
  }
```

The concrete representation maps every lexical constant to a full-form inflection table, computed by applying a smart paradigm to one or more forms of a word. Now consider the following English concrete lexicon segment for the above given abstract lexicon segment:

```
concrete DictEng of DictAbs = {
  lin boy_N = mkN "boy" ;
  lin lesson_N = mkN "lesson" ;
  lin tall_A = mkA "tall" ;
  lin learn_V2 = mkV2 (mkV "learn") ;
  }
```

Here, **mkN, mkA**, and **mkV** are smart paradigms. They take the canonical form of a word and produce full-form inflection tables. For example the paradigm **mkV** in the above code takes the infinitive form of the verb "learn" and produces its inflectional forms given in Table 5.1.

GF allows multiple concrete representations for a single abstract representation. In this setting, the abstract representation acts as an interlingua and together with the parallel concrete representations results into a multi-lingual translation lexicon. For example, we can have the following parallel Hindi representation of the above given abstract representation:

```
concrete DictHin of DictAbs = {
  lin boy_N = mkN "larka:" ;
  lin lesson_N = mkN "sabak" ;
  lin tall_A = mkA "lamba:" ;
  lin learn_V2 = mkV2 (mkV "si:khna:") ;
  }
```

The API uses the same paradigm names for all languages as far as possible, although the inflection tables very from language to language. (The actual Hindi lexicon uses the Devanagari script encoded in UTF-8.)

5.3 Monolingual Lexicons

The first comprehensive mono-lingual lexicon for English reported in GF was based on the Oxford Advanced Learner's dictionary[3]. This lexicon has around 43000 lemmas. The Princeton Word-Net [Miller, 1995], on the other hand, provides a much bigger repository of English lexical data. We have extended our previous English lexicon to around 64700 entries by adding extra nouns, adjectives and adverbs from the Princeton WordNet data.

5.4 Multi-lingual Lexicons

As described previously, considering the abstract representation as an inter-lingua, it becomes possible to have multiple parallel concrete representations, which results into multi-lingual lexicons. Because words can have multiple senses, and it is often very hard to find one-to-one word mappings between languages, we develop two different types of multi-lingual lexicons: Uni-Sense and Multi-Sense. These lexicons have different purposes and uses given in the following subsections.

5.4.1 Uni-Sense Lexicons

In a uni-sense lexicon each source word is restricted to represent one particular sense of the word, and hence it becomes easier to map it to one particular word in the target language. These type of lexicons are useful for building domain specific NLP applications.

The abstract representation of our uni-sense lexicon has around 64700 words, which are based on the Oxford Advanced Learner's Dictionary, and the Princeton WordNet. During the development of the concrete lexicons a guiding principle was to reuse the existing lexical resources of these languages as much as possible. So, the parallel data for building Hindi lexicon was taken from the Hindi WordNet [Jha et al., 2001], and for Urdu, the data was extracted from different resources including Waseem Siddiqi's English-Urdu dictionary[4]. Table 5.2 gives statistics about the coverage of both lexicons.

[3]http://oald8.oxfordlearnersdictionaries.com/

[4]freely available for downloading and editing at

Lang	Abstract	Hindi	Urdu
Size	64700	33600	30000

Table 5.2: Lexicon Coverage Statistics

In situations, when there were one-to-many mappings from source-to-target language an obvious issue was the choice of a lexical entry. Currently for the English-Hindi lexicon we solved this issue by simply selecting the first Hindi word (essentially random) and later tailoring it to a particular domain. For the same reason, we have developed two different versions of the Hindi lexicon: One tailored to the Wall Street Journal (WSJ), and the other to the mathematics domain. However, it is important to mention that the tailoring was done during an experiment and the percentage of this tailoring is very low.

Another issue was that it can not always be guaranteed that the smart paradigm will predict the correct lexical paradigm for word-forms generation, and this can result in irregularities and/or bugs. In the case of Urdu, one such example is the noun (آدمی,'a:dmi:',man) versus (کرسی,'kursi:',chair). The Urdu smart paradigm (mkN) tries to guess the gender of a noun from its ending. Now, both these nouns end at 'ی', but (آدمی,'a:dmi:',man) is masculine while (کرسی,'kursi:',chair) is feminine. The default behavior will predict (آدمی,'a:dmi:',man) to be feminine and will result into bugs at the syntax level. Either such entries need to be corrected manually (i.e. in this case by replacing one argument mkN (i.e. mkN "آدمی") with the two arguments version (i.e. mkN "آدمی" masculine)), or we need some other automatic solution. Currently, we put such issues in the 'known-issues' list and leave them for future.

5.4.2 Multi-Sense Lexicons

A multi-sense lexicon is a more comprehensive lexicon and contains multiple senses of words and their translations to other languages. This type of lexicons can be used for open-domain tasks such as arbitrary text translation. These multi-sense lexicons have been developed using data from the Princeton WordNet and the Universal WordNet [de Melo and Weikum, 2009]. The original Princeton WordNet defines a set of word senses, and the Universal WordNet maps them to different languages. In this multilingual scenario, the Princeton WordNet senses can be seen as an abstract representation, while

http://www.scribd.com/doc/8509778/English-to-Urdu-Dictionary

the Universal WordNet mappings can be seen as concrete representation of those senses in different languages. This section briefly describes the experiment we did to build one abstract and multiple concrete GF lexicons for a number of languages including German, French, Finnish, Swedish, Hindi, and Bulgarian. The method is very general and can be used to build similar lexicon for other languages.

Abstract Lexicon

The Princeton WordNet data is distributed in the form of different database files. For each of the four lexical categories (i.e. noun, verb, adjective, and adverb), two files named 'index.pos' and 'data.pos' are provided, where 'pos' is either noun, verb, adj or adv. Each of the 'index.pos' files contains all words, including synonyms of the words, found in the corresponding part of speech category. Each of the 'data.pos' files contains data about unique senses belonging to the corresponding part of speech category. For our purposes, there were two possible choices to build an abstract representation of the lexicon:

1. To include all words of the four lexical categories, and also their synonyms (i.e. to build the lexicon from 'index.pos' files)

2. To include only unique senses of the four categories with one word per sense, but not the synonyms (i.e. to build the lexicon from the 'data.pos' files)

To better understand this difference, consider the words 'brother' and 'buddy'. The word 'brother' has five senses with sense offsets '08111676', '08112052', '08112961', '08112265' and '08111905' in the Princeton WordNet 1.7.1, while the word 'buddy' has only one sense with the sense offset '08112961'. Choosing option (1) means that we have to include the following entries in our abstract lexicon.

```
brother_08111676_N
brother_08112052_N
brother_08112961_N
brother_08112265_N
brother_08111905_N
buddy_08112961_N
```

We can see that the sense with the offset '08112961' is duplicated in the lexicon: once with the lemma 'brother' and then with the lemma 'buddy'. However, if we choose option (2), we end up with the following entries.

```
brother_08111676_N
brother_08112052_N
brother_08112265_N
brother_08111905_N
buddy_08112961_N
```

Since the file 'data.noun' lists the unique senses rather than the words, their will be no duplication of the senses. However, the choice has an obvious effect on the lexicon coverage, and depending on whether we want to use it as a parsing or as a linearization lexicon, the choice becomes critical. Currently, we choose option (2) for the following reason:

The Universal WordNet provides mappings for synsets (i.e. unique senses) but not for the individual synonyms of the synsets. If we choose option (1), as mentioned previously, we have to list all synonyms in our abstract representation. But, as translations are available only for synsets, we have to put the same translation against each of the synonym of the synset in our concrete representations. This will not gain anything but will increase the size of the lexicon and hence may have a negative impact on the processing speed.

Our abstract GF lexicon covers 91516 synsets out of around 111,273 synsets in the WordNet 1.7.1. We exclude some of the synsets with multi-word lemmas. We consider them as of syntactic categories, and hence deal with them at the syntax level. Here, we give a small segment of our abstract GF lexicon.

```
abstract LinkedDictAbs = Cat ** {

  fun consentaneous_00526696_A : A ;
  fun consecutive_01624944_A  : A ;
  fun consequently_00061939_Adv : Adv ;
  fun abruptly_00060956_Adv  : Adv ;
  fun consequence_09378924_N : N ;
  fun consolidation_00943406_N   : N ;
  fun consent_05596596_N  : N ;
  fun conservation_06171333_N : N ;
  fun conspire_00562077_V : V ;
  fun sing_01362553_V2  : V2 ;
  ........
  ........
 }
```

The first line in the above given code states that the module 'LinkedDictAbs' is an abstract representation (note the keyword 'abstract'). This module extends (achieved by '**' operator) another module labeled 'Cat[5]', which in this case, has definitions for the morphological categories 'A', 'Adv', 'N' and 'V'. These categories correspond to the 'adjective', 'adverb', 'noun', and 'verb' categories in the WordNet respectively. However, note that in GF resource grammars we have a much fine-grained morphological division for verbs. We sub-categorize them according to their valencies i.e 'V' is for intransitive, and 'V2' for transitive verbs (we refer to [Bringert et al., 2011] for more details on this division).
Each entry in this module is of the following general type:

```
fun lemma_senseOffset_t : t ;
```

Keyword 'fun' declares each entry as a function of the type 't'. The function name is composed of lemma, sense offset and a type 't', where lemma and sense offset are same as in the Princeton WordNet, while 't' is one of the morphological types in GF resource grammars. This abstract representation will serve as a pivot for all concrete representations, which are described next.

Concrete Lexicons

We build the concrete representations for different languages using the translations obtained from the Universal WordNet data and GF morphological paradigms [Détrez and Ranta, 2012, Bringert et al., 2011]. The Universal WordNet translations are tagged with a sense offset from WordNet 3.0[6] and also with a confidence score. As an example consider the following segment from the Universal WorNet data, showing German translations for the noun synset with offset '13810818' and lemma 'rest' (in the sense of 'remainder').

```
n13810818 Rest             1.052756
n13810818 Abbrand          0.95462
n13810818 Ruckstand      0.924376
n13810818 Restbetrag      0.662388
n13810818 Restauflage      0.446788
n13810818 Restglied      0.446788
```

[5]This module has definitions of different morphological and syntactic categories in the GF resource grammar library.

[6]In our concrete lexicons we match them to WordNet 1.7.1 for the reason that in our recent experiments, we are using these lexicons to build an arbitrary text translator. This translation system is using an external Word-Sense disambiguation tool, which is based on WordNet 1.7.1. However, this can easily be mapped back and forth to the other WordNet versions.

```
n13810818 Restbestand     0.446788
n13810818 Residuum      0.409192
```

Each entry is of the following general type:

```
posSenseOffset translation confidence-score
```

In cases, where we have more than one candidate translations for the same sense (as in the case of 'rest'), we select the best one (i.e. with the maximum confidence score) and put it in the concrete grammar. Next, we give a small segment from the German concrete lexicon for the above given abstract lexicon segment.

```
concrete LinkedDictGer of LinkedDictAbs = CatGer ** open
 ParadigmsGer, IrregGer,Prelude in  {

  lin consentaneous_00526696_A =  mkA "einstimmig" ;
  lin consecutive_01624944_A =  mkA "aufeinanderfolgend" ;
  lin consequently_00061939_Adv =  mkAdv "infolgedessen" ;
  lin abruptly_00060956_Adv =  mkAdv "gech" ;
  lin consequence_09378924_N =  mkN "Auswirkung" ;
  lin consolidation_00943406_N =  mkN "Konsolidierung" ;
  lin consent_05596596_N =  mkN "Zustimmung" ;
  lin conservation_06171333_N =  mkN "Konservierung" ;
  lin conspire_00562077_V = mkV "anzetteln" ;
  lin sing_01362553_V2 = mkV2 (mkV "singen" ) ;
  ......
  ......
 }
```

The first line declares 'LinkedDictGer' to be the concrete representation of the previously defined abstract representation (note the keyword 'concrete' at the start of the line). Each entry in this representation is of the following general type:

```
lin lemma_senseOffset_t = paradigmName "translation" ;
```

Keyword 'lin' declares each entry to be a linearization of the corresponding function in the abstract representation. 'paradigmName' is one of the morphological paradigms defined in the 'ParadigmsGer' module. So in the above code, 'mkA', 'mkAdv', 'mkN', 'mkV' and 'mkV2' are the German morphological paradigms[7] for different lexical categories (i.e. 'adjective', 'adverb',

[7]See [Bringert et al., 2011] for more details on these paradigms

'noun', 'intransitive verb', and 'transitive verb' respectively). 'translation' in double quotes is the best possible translation obtained from the Universal WordNet. This translation is passed to a paradigm as a base word, which then builds a full-form inflection table.

Concrete lexicons for all other languages were developed using the same procedure. Table 5.3 gives statistics about the coverage of these lexicons. The Finnish lexicon was later revised and extended using the data from the Finnish WordNet [Lindén and Carlson, 2010].

Language	Number of entries
Abstract	91516
German	49439
French	38261
Finnish	27673
Swedish	23862
Hindi	16654
Bulgarian	12425

Table 5.3: Lexicon Coverage Statistics

Part III

Applications

Chapter 6

Computational evidence that Hindi and Urdu share a grammar but not the lexicon

This chapter describes the mechanical development of a Hindi resource grammar starting from an Urdu resource grammar. It also gives a detailed analysis of the difference between Hindi and Urdu resource grammars at different levels: morphology, syntax, and script. At the end, it describes the evaluation experiments that resulted in the conclusions that Hindi and Urdu lexicons differ in 18% of the basic words, in 31% of tourist phrases, and in 92% of school mathematics terms.

The layout has been revised.

Abstract: Hindi and Urdu share a grammar and a basic vocabulary, but are often mutually unintelligible because they use different words in higher registers and sometimes even in quite ordinary situations. We report computational translation evidence of this unusual relationship (it differs from the usual pattern, that related languages share the advanced vocabulary and differ in the basics). We took a GF resource grammar for Urdu and adapted it mechanically for Hindi, changing essentially only the script (Urdu is written in Perso-Arabic, and Hindi in Devanagari) and the lexicon where needed. In evaluation, the Urdu grammar and its Hindi twin either both correctly translated an English sentence, or failed in exactly the same grammatical way, thus confirming computationally that Hindi andUrdu share a grammar. But the evaluation also found that the Hindi and Urdu lexicons differed in 18% of the basic words, in 31% of tourist phrases, and in 92% of school mathematics terms.

Keywords: Grammatical Framework, Resource Grammars, Application Grammars

6.1 Background facts about Hindi and Urdu

Hindi is the national language of India and Urdu that of Pakistan, though neither is the native language of a majority in its country.

'Hindi' is a very loose term covering widely varying dialects. In this wide sense, Hindi has 422 million speakers according to [Census-India, 2001]. This census also gives the number of native speakers of 'Standard Hindi' as 258 million. Official Hindi now tends to be Sanskritised, but Hindi has borrowed from both Sanskrit and Perso-Arabic, giving it multiple forms, and making Standard Hindi hard to define. To complete the 'national language' picture, note that Hindi is not understood in several parts of India [Agnihotri, 2007], and that it competes with English as lingua franca.

It is easier, for several reasons, to talk of standard Urdu, given as the native language of 51 million in India by [Census-India, 2001], and as that of 10 million in Pakistan by [Census-Pakistan, 1998]. Urdu has always drawn its advanced vocabulary only from Perso-Arabic, and does not have the same form problem as Hindi. It is the official language and lingua franca of Pakistan, a nation now of 180 million, though we note that Urdu's domination too is contested, indeed resented in parts of the country [Sarwat, 2006].

Hindi and Urdu 'share the same grammar and most of the basic vocabulary of everyday speech' [Flagship, 2012]. This common base is recognized, and known variously as 'Hindustani' or 'Bazaar language' [Chand, 1944,

Naim, 1999]. But, 'for attitudinal reasons, it has not been given any status in Indian or Pakistani society' [Kachru, 2006]. Hindi-Urdu is the fourth or fifth largest language in the world (after English, Mandarin, Spanish and perhaps Arabic), and is widely spoken by the South Asian diaspora in North America, Europe and South Africa.

6.1.1 History: Hindustani, Urdu, Hindi

From the 14th century on, a language known as Hindustani developed by assimilating into Khari Boli, a dialect of the Delhi region, some of the Perso-Arabic vocabulary of invaders. Urdu evolved from Hindustani by further copious borrowing from Persian and some Arabic, and is written using the Perso-Arabic alphabet. It dates from the late 18th century. Hindi, from the late 19th century, also evolved from Hindustani, but by borrowing from Sanskrit. It is written in a variant of the Devanagari script used for Sanskrit.

But the Hindi/Urdu has base retained its character: 'the common spoken variety of both Hindi and Urdu is close to Hindustani, i.e., devoid of heavy borrowings from either Sanskrit or Perso-Arabic' [Kachru, 2006].

6.1.2 One language or two?

Hindi and Urdu are 'one language, two scripts', according to a slogan over the newspaper article [Joshi, 2012]. The lexicons show that neither Hindi nor Urdu satisfies that slogan. Hindustani does, by definition, but is limited to the shared part of the divergent lexicons of Hindi and Urdu.

[Flagship, 2012] recognizes greater divergence: it says Hindi and Urdu 'have developed as two separate languages in terms of script, higher vocabulary, and cultural ambiance'. Gopi Chand Narang, in his preface to [Schmidt, 2004] stresses the lexical aspect: 'both Hindi and Urdu share the same Indic base ... but at the lexical level they have borrowed so extensively from different sources (Urdu from Arabic and Persian, and Hindi from Sanskrit) that in actual practice and usage each has developed into an individual language'.

But lexical differences are not quite the whole story. [Naim, 1999] lists several subtle morphological differences between Hindi and Urdu, and some quite marked phonological ones. Most Hindi speakers cannot pronounce the Urdu sounds that occur in Perso-Arabic loan words: `q` (unvoiced uvular plosive), `x` (unvoiced velar fricative), `G` (voiced velar fricative), and some final consonant clusters, while Urdu speakers replace the `n̤` (retroflex nasal) of Hindi by `n`, and have trouble with many Hindi consonant clusters.

Naim does not think it helps learners to begin with Hindi and Urdu together. Those who seek a command of the written language, he says, might as well learn the conventions exclusive to Urdu from the beginning.

Thus there are many learned and differing views on whether Hindi and Urdu are one or two languages, but nothing has been computationally proved, to the best of our knowledge. Our work demonstrates computationally that Hindi and Urdu share a grammar, but that the lexicons diverge hugely beyond the basic and general registers. Our as yet first experiments already give preliminary estimates to questions like 'How much do Hindi and Urdu differ in the lexicons?'.

Overview Section 6.2 describes Grammatical Framework, the tool used in this experiment, and Section 6.3 lists what we report. Section 6.4 describes the Hindi and Urdu resource grammars, some differences between them, and how we cope with these differences. Section 6.5 presents the general and domain-specific lexicons used in this experiment. Evaluation results are given at the ends of Sections 6.4 and 6.5. Section 6.6 provides context and wraps up.

This paper uses an IPA style alphabet, with the usual values and conventions. Retroflexed sounds are written with a dot under the letter; ʈ, ɖ, and ɽ (a flap) are common to Hindi and Urdu, while ɳ and ʂ occur in Sanskritised Hindi (though many dialects pronounce them n and š). The palatalised spirant š and aspirated stops, shown thus: k^h, are common to Hindi and Urdu. A macron over a vowel denotes a long vowel, and ~, nasalisation. In Hindi and Urdu, e and o are always long, so the macron is dropped. Finally, we use ñ to mean the nasal homorganic with the following consonant.

6.2 Background: Grammatical Framework

Grammatical Framework (GF) [Ranta, 2004] is a grammar formalisim tool based on Martin Löf's [Martin-Löf, 1982] type theory. It has been used to develop multilingual grammars that can be used for translation. These translations are not usually for arbitrary sentences, but for those restricted to a specific domain, such as tourist phrases or school mathematics.

6.2.1 Resource and Application Grammars in GF

The sublanguages of English or Hindi, say, that deal with these specific domains are described respectively by the (English or Hindi) *application grammars* Phrasebook (Caprotti et al 2010, [Ranta et al., 2012] and MGL

[Caprotti and Saludes, 2012]. But the English Phrasebook and English MGL share the underlying English (similarly for Hindi). The underlying English (or Hindi) syntax, morphology, predication, modification, quantification, etc., are captured in a common general-purpose module called a *resource grammar*.

Resource grammars are therefore provided as software libraries, and there are currently resource grammars for more than twenty five languages in the GF resource grammar library [Ranta, 2009b]. Developing a resource grammar requires both GF expertise and knowledge of the language. Application grammars require domain expertise, but are free of the general complexities of formulating things in English or Hindi. One might say that the resource grammar describes how to speak the language, while the application grammar describes what there is to say in the particular application domain.

6.2.2 Abstract and Concrete Syntax

Every GF grammar has two levels: abstract syntax and concrete syntax. Here is an example from Phrasebook.

1. Abstract sentence:
   `PQuestion (HowFarFrom (ThePlace Station)(ThePlace Airport))`
2. `Concrete English sentence: How far is the airport from the station?`
3. `Concrete Hindustani sentence: sṭešan se havāī aḍḍā kitnī dūr hæ?`
   (स्टेशन से हवाई अड्डा कितनी दूर है? , اسٹیشن سے ہوائی اڈا کتنی دور ہے؟)
4. `Hindustani word order: station from air port how-much far is?`

The abstract sentence is a tree built using functions applied to elements. These elements are built from categories such as questions, places, and distances. The concrete syntax for Hindi, say, defines a mapping from the abstract syntax to the textual representation in Hindi. That is, a concrete syntax gives rules to linearize the trees of the abstract syntax.

Examples from MGL would have different abstract functions and elements. In general, the abstract syntax specifies what categories and functions are available, thus giving language independent semantic constructions.

Separating the tree building rules (abstract syntax) from the linearization rules (concrete syntax) makes it possible to have multiple concrete syntaxes

for one abstract. This makes it possible to parse text in one language and output it in any of the other languages.

Compare the above tree with the resource grammar abstract tree for "How far is the airport from the station?" to see the difference between resource and application grammars:

```
PhrUtt NoPConj (UttQS (UseQCl (TTAnt TPres ASimul) PPos
(QuestIComp (CompIAdv (AdvIAdv how_IAdv far_Adv)) (DetCN
(DetQuant DefArt NumSg) (AdvCN (UseN airport_N) (PrepNP
from_Prep (DetCN (DetQuant DefArt NumSg) (UseN station_N)
))))))) NoVoc
```

6.3 What we did: build a Hindi GF grammar, compare Hindi/Urdu

We first developed a new grammar for Hindi in the Grammatical Framework (GF) [Ranta, 2011] using an already existing Urdu resource grammar [Shafqat et al., 2010]. This new Hindi resource grammar is thus the first thing we report, though it is not in itself the focus of this paper.

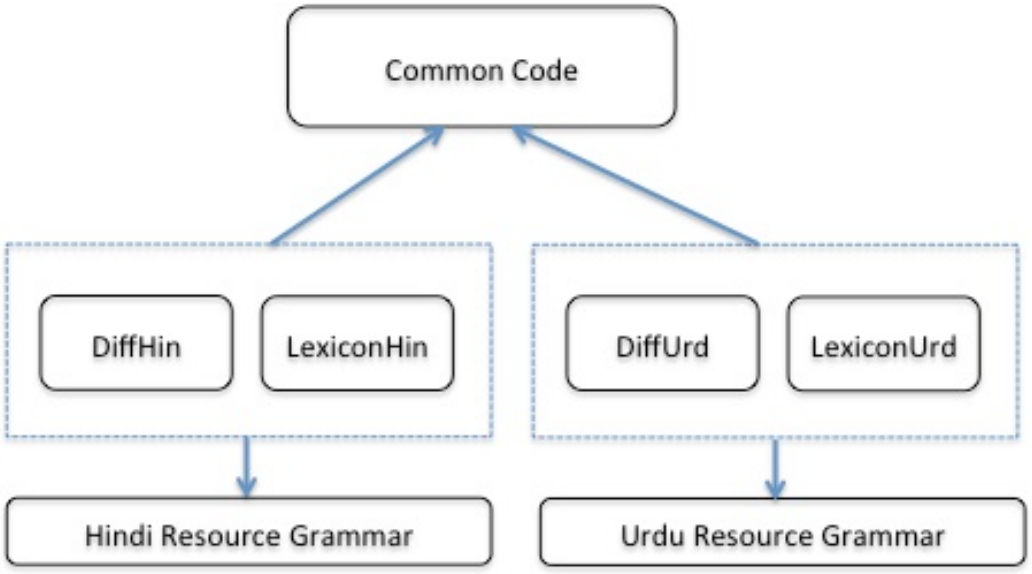

Figure 6.1: Hindi/Urdu Functor.

We used a functor style implementation to develop Hindi and Urdu resource grammars, which makes it possible to share commonalities between

two grammars. Figure 6.1 gives a picture of this implementation style. Most of the syntactic code resides in the 'common code box', and the minor syntactical differences (discussed in Section 6.4) are placed in each of the 'DiffLang box'. Each resource grammar has its own lexicon. This mechanically proves that Hindi and Urdu share a grammar and differ almost only in the lexicons.

We evaluated our claim by (1) porting two application grammars to Hindi and Urdu: MGL, a mathematical grammar library for school mathematics [Caprotti and Saludes, 2012], and a Phrasebook of tourist sentences [Ranta et al., 2012], (2) randomly producing 80 abstract trees (40 from each of the Phrasebook, and MGL), (3) linearizing them to both Hindi and Urdu, and finally checking them either for correctness, or badness (see Section 6.6 for results).

6.4 Differences between Hindi and Urdu in the Resource Grammars

We started from the script based GF resource grammar for Urdu, and adapted it for Hindi almost entirely just by re-coding from Urdu to Hindi script. A basic test vocabulary accompanies the resource grammars, and this was changed as needed: it turned out that Hindi and Urdu differ up to 18% even in this basic vocabulary. Section 6.5 deals with the application lexicons.

We do not give any implementation details of these resource grammars in this paper, as the interesting bits can be found in [Shafqat et al., 2010]. But we describe below resource level differences between Hindi/Urdu, and strategies to deal with them.

6.4.1 Morphology

Every GF resource grammar provides a basic test lexicon of 450 words, for which the morphology is programmed by special functions called lexical paradigms. Our Hindi morphology simply takes the existing Urdu morphology and re-codes it for the Devanagari script. Lexical differences mean that the morphologies are not identical; e.g., Hindi sometimes uses a simple word where Urdu has a compound word, or vice-versa. But there are no patterns that occur in only one of the languages, so the test lexicon for Hindi works with few problems.

We could in principle implement the subtle morphological differences noted in [Naim, 1999], but we ignored them. That these differences are minor is shown by the fact that our informants find the resulting Hindi entirely normal.

6.4.2 Internal Representation: Sound or Script?

The translation of "How far is the airport from the station?" was written in IPA, representing the sound of the Hindi/Urdu. It sounds identical in the two languages, and thus we could label it 'Hindustani'. An obvious approach to writing grammars for Hindi/Urdu from scratch would be to represent the languages internally by sound, so that we would get just one grammar, one shared lexicon, and differentiated lexicons only for those words that sound different in Hindi and Urdu. For output, we would then map the IPA to the Hindi or Urdu script.

But we were starting from [Shafqat et al., 2010], which uses an internal representation based on written Urdu. It would be a fair sized task to redo this in terms of speech, though the result would then be immediately re-usable for Hindi and might also help capture similarities to other South Asian languages. We reserve this re-modelling for future work.

So, in the present work, we changed the Urdu grammar to a Hindi grammar merely by replacing written Urdu by written Hindi. This script change was also done for the basic lexicon, though here some words were indeed different even spoken. Our parallel grammars therefore give no indication that Hindi and Urdu often sound identical.

One compensating advantage is that script-based representations avoid spelling problems. Hindi-Urdu collapses several sound distinctions in Persian, Arabic and Sanskrit. A phonetic transcription would not show these collapsed distinctions, but the orthography does, because Urdu faithfully retains the spelling of the original Perso-Arabic words while representing Sanskrit words phonetically, while Hindi does the reverse. Each language is faithful to the sources that use the same script. We see that it will not be entirely trivial to mechanically go from a phonetic representation to a written one.

Obviously, the more the Hindi and Urdu lexicons overlap, the more the wasted effort in the parallel method. But as we shall see, the lexicons deviate from each other quite a bit. We have designed an augmented phonetic representation that keeps track of spelling, for use in a remodelled grammar.

6.4.3 Idiomatic, Gender and Orthographic Differences

In addition to spelling, Hindi and Urdu also have orthographic differences, not often remarked. Indeed some apparently grammatical differences result from in fact idiomatic, gender or orthographic differences.

For example, the lexicon might translate the verb "to add" as "`joṛnā`" in Hindi, and as "`jame karnā`" in Urdu. The imperative sentence "add 2 to 3"

would then be rendered "do ko tīn se joṛo" in Hindi, and "do ko tīn mẽ jame karo" in Urdu. But the choice between the post-positions "se" and "mẽ" is determined not by different grammars for Hindi and Urdu, but by the post-positional idiom of the chosen verb, "joṛnā" or "jame karnā", as can be seen because either sentence works in either language.

A gender difference appears with "war", rendered in Urdu as "laṛāī" (fem.). This word works in Hindi as well, but has more a connotation of "battle", so we chose instead "sañgʰarṣ" (masc.). The shift from feminine to masculine is driven by the choice of word, not language.

Orthographic differences next. "He will go" is "vo jāegā" in both languages; in writing, (वह जाएगा, وہ جائے گا), the final "gā" (गा, گا) is written as a separate word in Urdu but not in Hindi. Similarly, "we drank tea" is "hamne cāy pī" in both languages, but in writing, (हमने चाय पी, ہم نے چائے پی), the particle "ne" (ने, نے) is written as a separate word in Urdu but not in Hindi.

These differences were handled by a small variant in the code, shown below. To generate the future tense for Urdu, the predicate is broken into two parts: finite (fin) and infinite (inf). The inf part stores the actual verb phrase (here "jāe"), and the fin part stores the copula "gā" as shown below.

```
VPFut=>fin=(vp.s! VPTense VPFutr agr).fin; inf=(vp.s!
VPTense VPFutr agr).inf
```

For Hindi, these two parts are glued to each other to make them one word. This word is then stored in the inf part of the predicate and the fin part is left blank as shown below.

```
VPFut=>fin=[]; inf=Prelude.glue ((vp.s! VPTense VPFutr
agr).inf) ((vp.s! VPTense VPFutr agr).fin)
```

Similarly in the ergative "hamne cāy pī" ("we drank tea"), Urdu treats "ham" and "ne" as separate words, while Hindi makes them one. We used for Urdu, `NPErg => ppf ! Obl ++ "ne"` and for Hindi, `NPErg => glue (ppf ! Obl) "ne"`.

6.4.4 Evaluation and Results

With external informants

As described earlier, we randomly generated 80 abstract trees (40 from each of the Phrasebook, and MGL), linearized them to both Hindi and Urdu. These linearizations were then given to three independent informants.

They evaluated the Hindi and Urdu translations generated by our grammars. The informants found 45 sentences to be correct in both Hindi and Urdu. The other sentences were found understandable but failed grammatically - in exactly the same way in both Hindi and Urdu: nothing the informants reported could be traced to a grammatical difference between Hindi and Urdu. For this paper, the point is that all 80 sentences, the badly translated as well as the correctly translated, offer mechanical confirmation that Hindi and Urdu share a grammar.

We note for the record that the 35 grammatical failures give a wrong impression that the grammar is only "45/80" correct. In fact the grammar is much better: there are only a handful of distinct known constructs that need to be fixed, such as placement of negation and question words, but these turn up repeatedly in the evaluation sentences.

A result that has not been the focus of this paper is that we greatly improved the Urdu grammar of [Shafqat et al., 2010] while developing the Hindi variant. Errors remain, as noted above.

With internal informants

The second author is a native Urdu speaker, while the first speaks Hindi, though not as a native. With ourselves as internal informants, we could rapidly conduct several more extensive informal evaluations. We looked at 300 Phrasebook sentences, 100 MGL sentences, and 100 sentences generated directly from the resource grammars. We can confirm that for all of these 500 English sentences, the corresponding Urdu and Hindi translations were understandable and in conformance with Urdu and Hindi grammar (barring the known errors noted by the external informants).

We note particularly that randomly generated MGL sentences can be extremely involuted, and that the Hindi and Urdu translations had the same structure in every case.

6.5 The Lexicons

As we noted in Section 6.1, Urdu has a standard form, but Hindi does not, though official Hindi increasingly tends to a Sanskritised form. Hindustani itself counts as 'Hindi', and is a neutral form, but has only basic vocabulary, a complaint already made in [Chand, 1944]. So to go beyond this, Hindi speakers have to choose between one of the higher forms. Elementary mathematics, for example, can be done in Hindustani or in Sanskritised Hindi, attested by the NCERT books [NCERT, 2012], or in English-ised Hindi, which can be

heard at any high school or university in the Hindi speaking regions.

We arbitrated the choice of Hindi words thus: when we had sources, such as the NCERT mathematics books or a government phrase book, we used those. Otherwise, we used [Snell and Weightman, 2003] and [Jha et al., 2001] to pick the most popular choices.

6.5.1 The general lexicon

Out of 350 entries, our Hindi and Urdu lexicons use the same word in 287 entries, a fraction of 6/7 which can easily be changed by accepting more Urdu words as Hindi' or by avoiding them. We note in passing that the general lexicon is any case often tricky to translate to Hindi-Urdu, as the cultural ambience is different from the European one where GF started, and which the test lexicon reflects. Many words ("cousin", "wine", etc.) have no satisfactory single equivalents, but these lexical items still help to check that the grammars work.

6.5.2 The Phrasebook lexicon

This lexicon has 134 entries, split into 112 words and 22 greetings. For 92 of the words, the Hindi and Urdu entries are the same; these include 42 borrowings from English for names for currencies, (European) countries and nationalities,and words like "tram" and "bus". So Hindi and Urdu share 50 of 70 native words, but differ on 20, including days of the week (except Monday, "`somvār`" in both Hindi and Urdu). The greetings lexicon has 22 entries, most of which are hard to translate. "Good morning" etc. can be translated though they are often just "hello" and "bye". Greetings are clearly more culture dependent: Hindi and Urdu differ in 17 places.

An example not in the Phrasebook drives home the point about greetings: airport announcements beginning "Passengers are requested ..." are rendered in Hindi as "`yātriyõ se nivedan hæ` ..." (यात्रियों से निवेदान है) and in Urdu as "`musāfirõ se guza:riš kī jātī hæ` ..." (مسافروں سے گزارش کی جاتی ہے), which suggests that Hindi and Urdu have diverged even in situations almost tailored for 'Bazaar Hindustani'!

6.5.3 The Mathematics lexicon

Our MGL lexicon, for use with high school mathematics, has 260 entries. Hindi and Urdu differ on 245 of these. The overlapping 15 include function words used in a technical mathematical sense, "such that", "where", and so on.

As examples of the others, here are some English words with their Hindi and Urdu equivalents in parentheses: perpendicular (lañb लंब, amūd عمود), right-angled (samkoṇ समकोण, qāyam zāvī قائم زاوی), triangle (trib^{h}uj त्रिभुज, mašallaš مثلث), hypotenuse (karṇ कर्ण , vitar وتر), vertex (šīrṣ शीर्ष, rās راس).

This total divergence comes about because Urdu borrows mathematical terms only from Perso-Arabic, and Hindi, only from Sanskrit. There would be more overlap in primary school, where Hindi uses more Hindustani words, but the divergence is already complete by Class 6. The parallel English, Hindi and Urdu texts [NCERT, 2012], from which we got the list above, show that the grammar of the Hindi and Urdu sentences continue to be identical modulo lexical changes, even when the lexicons themselves diverge totally.

Since it often happens in mathematics that every Hindi content word is different from its Urdu counterpart, the languages are mutually unintelligible. Even function words can differ. Either "yadi" or "agar" can mean "if" in Hindi, but the Sanskrit "yadi" is often chosen for reasons of stylistic unity with the Sanskrit vocabulary. Urdu never uses "yadi".

More on Hindi mathematical terms

Our Hindi words were taken mostly from the NCERT books, which particularly in the later classes use Sanskritised Hindi. They make good use of the regular word-building capacity of Sanskrit. For example, "to add" is "joṛnā" in the lower classes, but "addition" becomes "yog" in the higher classes. This allows constructs like (yogātmak, additive), which is like (guṇātmak, multiplicative), (b^{h}āgātmak, divisive) and so on.

One might think the NCERT books overly Sanskritised, but it is hard to find other solutions, short of massive code switching between English and Hindi. NCERT books are widely used all over India. We have no sales figures for the NCERT mathematics books in Hindi, but there are not many widely available alternatives. If Hindi is to become a language for mathematics, these books might be a major lexical source.

6.5.4 Contrast: the converging lexicons of Telugu/Kannada

Hindi and Urdu make a very unusual pair, agreeing so completely at the base and diverging so much immediately after. Related languages usually go the other way. An example is the pair Telugu/Kannada, two South Indian languages.

Telugu/Kannada do not share a base lexicon, and so are mutually unintelligible for everyday use, unlike Hindi/Urdu.

But at higher registers, where Hindi/Urdu diverge, Telugu/Kannada converge. So where a Hindi speaker listening to technical Urdu would understand the grammar but not the content words, the Telugu speaker listening to technical Kannada would recognise all the content words but not the grammar.

For mathematics, Telugu/Kannada use a Sanskrit-based lexicon essentially identical to that of Hindi. We do not list the exact Telugu and Kannada versions, but do note that the convergence Hindi-Telugu-Kannada would be improved by deliberate coordination. For completeness, we mention that a smaller part of the higher vocabulary, mostly administrative terms, is shared with Urdu.

Further,Telugu/Kannada are in fact grammatically close, so a Telugu speaker who knows no Kannada would need only a brief reminder of grammar and a basic lexicon to read mathematics in Kannada—the mathematical terms would be familiar. A hypothetical "Scientific Kannada for Telugu Speakers" need only be a slim volume. It is the general reading in Kannada that would need a bigger lexicon. This parallels the situation of an English speaking scientist trying to read French—the scientific reading is easier!

But for a Hindi-speaking scientist trying to read Urdu, it is the everyday texts that are easier, not the scientific ones.

6.5.5 Summary of lexical study

Our figures suggest that everyday Hindi and Urdu share 82% of their vocabulary, but this number drops if we move to a specific domain: for tourist phrases, to 69%, and for very technical domains, such as mathematics, to a striking 8%.

An English speaker who knows no mathematics might hear mathematics in English as built of nonsense words that function recognizably as nouns, adjectives, verbs and so on. This is how mathematics in Urdu would sound to a Hindi speaking mathematician (and the other way around), even though Hindi and Urdu share a base lexicon and the grammar.

The mathematics lexicons of Hindi, Telugu and Kannada suggest that a Sanskrit based vocabulary makes a powerful link across India. That vocabulary also makes Urdu the odd language out amongst Indian languages, despite its close relation to Hindi.

6.6 Discussion

Our results confirm that Hindi and Urdu share a grammar, but differ so much in vocabulary (even for travel and primary school) that they are now different languages in any but the most basic situation. With the various linguistic, cultural and political factors obtaining in India and Pakistan, a good guess is that the languages will diverge further.

A regular Sanskrit base for Hindi technical terms would cement this divergence from Urdu, but would give Hindi a more usual convergent relationship with other Indian languages, differing at the everyday level but coming together at higher registers. Indeed this situation might argue for Sanskritised Hindi as a national language, because for non-native Indian speakers this may be easier to understand than Hindi with more Perso-Arabic words.

[Paauw, 2009] says "Indonesia, virtually alone among post-colonial nations, has been successful at promoting an indigenous language as its national language." Pakistan may have similarly solved its national language problem, with a parallel situation of Urdu being the native language of a minority. A difference is that Urdu already has rich lexical and word-building resources, whereas Bahasa Indonesia did not. So the *Istilah* committee has over the decades standardised hundreds of thousands of terms. India does not need that many new terms, since it too has a rich shared lexical resource in Sanskrit, one that moreover has tremendous word-building capacity. But a standardising committee may help, since often the same Sanskrit word is used in different ways in different Indian languages. A standard pan-Indian lexicon for technical terms would allow for ease of translation, and might spur the usabilty of all Indian languages for science and technology.

Future Work

We hope to develop our Phrasebook and MGL tools, aiming for practical use. We also need to fix the remaining errors in our grammars, to do with continuous tenses, word order for some questions and negations, and the translation of English articles. Fixing these might be non-trivial. We have stated two other goals, to rebuild our resource grammars on a phonetic basis, and to do a progressive mathematics lexicon. We have started work on this last, which we believe will show an increasing divergence between Hindi and Urdu as we go to higher classes. The NCERT books are available in both Hindi and Urdu, so we have a ready made source for the lexicons.

Currently, popular articles and TV programs that need advanced vocabulary (e.g., music competitions or political debates) in Hindi take the terms needed from English, Urdu and Sanskrit sources, though these elements sit

uncomfortably together, at least as of now. More examples are worth studying.

Acknowledgements

We thank our informants Anurag Negi, Vinay Jethava, and Azam Sheikh Muhammad for their painstaking comments.

Our paper originally used French and English as an example of the usual relationship: shared technical vocabulary but differing everyday words. We thank one of our anonymous referees for pointing out that we should rather take a pair closer to home - they suggested Malay-Indonesian [Paauw, 2009], but we chose Telugu-Kannada both because the first author speaks these and because we can link them to Hindi via Sanskrit.

Chapter 7

Application Grammars

As described in previous chapters, there are two types of GF grammars: *Resource Grammars* and *Application Grammars*. The resource grammars can be used as libraries to build the application grammars. In this chapter, we describe how we have added support for different Indo-Iranian languages in three already existing GF application grammars: the MOLTO Phrasebook, the Mathematics Grammar Library (MGL), and the Attempto Controlled English (ACE) grammar.

7.1 The MOLTO Phrasebook

The MOLTO Phrasebook [Ranta et al., 2012] is a multilingual application grammar that can translate touristic phrases between 20 natural languages. This application grammar shows practically, how one can use the GF resource grammar library to quickly develop a multilingual application grammar for high quality translations. The grammar uses the GF's core concept of one abstract and multiple parallel concrete grammars for multilingualism, which in principle does not put any limit on the number of languages it can support. Initially the grammar supported 14 languages, but now this number has reached to 20. In this section, we show how one can add support for new languages using the respective language resource grammars.

The Phrasebook abstract grammar has a total of 42 categories and 290 functions. Out of these functions, 160 are lexical constants or canned phrases, while the remaining ones are syntactical constructions. To add support for a new language means to write the concrete representation of those categories and functions defined in the abstract grammar. Initially this looks like a hard task, but actually it is a lot simpler than it looks for the following reasons:

1. The linearizations of around 34 categories, and 88 functions are shared among different languages using *functors*. This means these linearizations need not to be rewritten for a new language. Few of these linearizations are:

   ```
   lincat
       Phrase = Text ;
       Greeting = Text ;
       Sentence = S ;
   lin
   -- Where is a hospital
   WherePlace place = mkQS (mkQCl where_IAdv place.name) ;
   -- Where are you
   WherePerson person = mkQS (mkQCl where_IAdv person.name) ;
   ```

   However, this does not always work and there is a possibility that some of those implementations are language-dependent. In that case, there is a way to exclude such implementation from the shared code and override them with the language-dependent versions.

2. There are many constructions which might need only the lexical replacements. For example consider the following implementation:

```
-- I know/I don't know/Does she know
AKnow p = mkCl p.name (mkV "know") ;
```

For any new language, replacing the lexical constant ”know” with the language-specific string might work. At least for Urdu the following lexical replacement worked:

```
AKnow p = mkCl p.name (mkV "جاننا") ;
```

3. The implementation of many of the other language-dependent syntactical constructions is possible using the resource grammar library (RGL) functions, which means these implementations need not to be written from scratch and RGL can assist.

Despite the above mentioned reasons, adding support for a new language might not be that straightforward. The reason is that there are many syntactical constructions (e.g. predication rules, question constructions etc.), which are language dependent and hence can not be shared, and for which simple lexical replacements are not enough. Sometimes, it might require a considerable effort to get them right. Consider the following construction:

```
QWhatAge : Person -> Question ;     -- how old are you
```

This is a typical example of a language-dependent construction and different languages treat it differently. In Urdu this construction will be treated in the sense of (”how much age you have”,”آپ کی عمر کتنی ہے”), and is implemented as follows:

```
QWhatAge p = mkQS (mkQCl howMuch_IAdv (mkNP p (mkN "عمر"
   feminine))) ;
```

The lexical paradigm ’mkN’ takes the lemma and the gender of the noun (عمر,o:mar,age) as arguments and builds a noun. The resource grammar API’s function `mkNP` takes the ’Person’ and the noun (عمر,o:mar,age) and builds a noun phrase. This noun phrase is then passed to the API’s function `mkQCl` together with the integrative adverb ’how much’, which builds the required clause. Even though the construction is very language dependent, the important point is that it is still possible to build it using the RGL functions and that the abstract syntax can be preserved.

We have added support for Hindi, Urdu and Persian in the Phrasebook using the corresponding language resource grammars and building the constructions from scratch where ever necessary. Figure 7.1 shows the touristic phrase ’where is a hospital’ and its translations in Hindi, Persian, and Urdu.

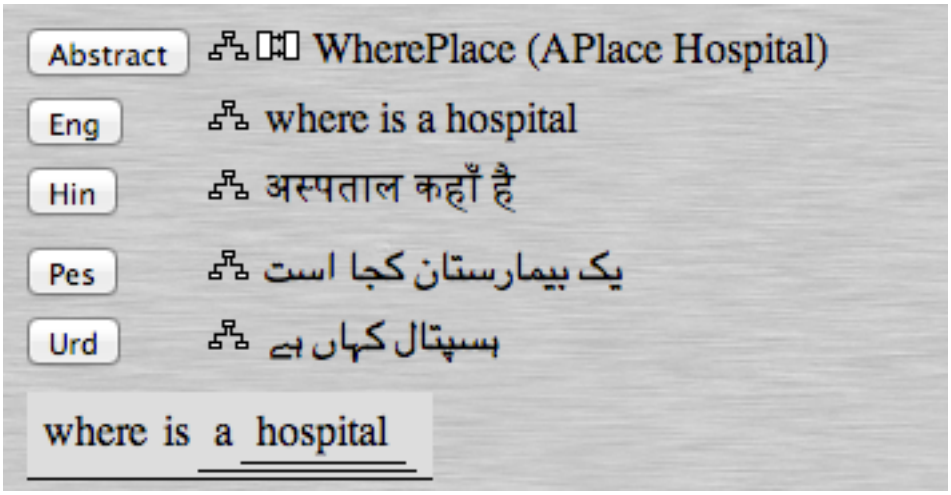

Figure 7.1: The Phrasebook Demo

7.2 MGL: The Mathematics Grammar Library

The mathematics grammar library (MGL) [Caprotti and Saludes, 2012] is an application grammar for translating mathematical expressions between different languages. The starting point of this library was the WebAlt[1] grammar, which has been extended and re-structured for better maintenance. There are many new constructions, and at the moment it has support for 13 natural languages including Bulgarian, Catalan, English, Finnish, French, German, Hindi, Italian, Polish, Romanian, Spanish, Swedish and Urdu. The grammar consists of approximately 36 categories, and around 275 functions covering a wide range of arithmetic constructions related to general arithmetic, algebra, geometry, logic, sets etc. The grammar also has a lexicon of around 250 arithmetic terms. We have added support for Urdu and Hindi in this library. Figure 7.2 shows the translation of the proposition 'the sum of X and Y is equal to Z' in 15 different languages including Hindi and Urdu. (visit http://www.grammaticalframework.org/demos/mathbar/ for a live demo)

7.3 The ACE Grammar

The Attempto[2] Controlled English (ACE) is a controlled natural language. It is a subset of natural English language and, by principle, is supposed to

[1] http://www.webalt.net/index_eng.html
[2] http://attempto.ifi.uzh.ch/site/

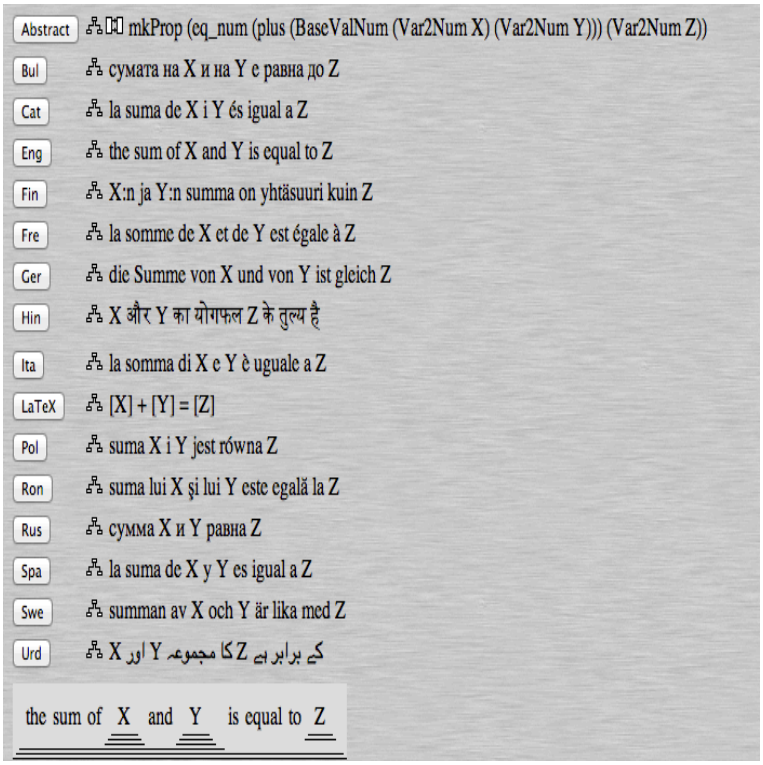

Figure 7.2: The MGL demo

be unambiguous. In an experiment [Angelov and Ranta, 2009], a GF application grammar was developed to show how GF can be used for controlled language implementations. The result of the experiment was a multilingual GF grammar, consisting of approximately 26 categories and 121 functions. This grammar can be used to translate the ACE documents between any of the five languages including English, French, Finnish, German, and Swedish. We have added support for Urdu, making it the sixth language in total. The grammar now has support for 15 languages, and have been used to implement a multilingual semantic wiki [Kaljurand and Kuhn, 2013].

Chapter 8

Towards an Arbitrary Text Translator

In this chapter, we describe our experiments to develop a wide-coverage arbitrary text translator. We propose different translation pipelines that use resource grammars and wide-coverage GF lexicons to translate arbitrary text. We also give baseline evaluation results of our system and compare it with Google translate. The main result is that with few limitations, our system can beat Google translate for under-resource languages like Urdu.

8.1 Introduction

Machine translation methods can be broadly classified into linguistic-based and non-linguistic-based translation paradigms [Dorr et al., 1998]. The linguistic based paradigms rely heavily on linguistic theories, and use well-grounded principles of morphology, syntax and semantics. Even though they are old-fashioned, they are closest to classical theories and philosophical view of natural languages. The rule-based, lexical-based, knowledge-based, and constraint-based machine translation systems lie in this category.

The non-linguistics based translation paradigms, on the other hand, rely on the availability of a vast amount of machine readable linguistic data (i.e. dictionaries, monolingual and bilingual parallel text corpora etc.), the processing capabilities of modern-age computers, and advanced machine learning algorithms. In the recent years, they have been in-fashion and hot research topics in the area of machine translation. As a result, there exist state of the art machine translation systems, like Google translate, which uses parallel text and statistical methods to translate text from one language to other. Example-based and neural-network-based machine translation systems are the other examples of the non-linguistic paradigm.

Both of the these paradigms have their own advantages and disadvantages. If we put them on an accuracy versus coverage graph, the linguistic-based approaches reside high on the quality axes, while the non-linguistic-based approaches reside high on the coverage axes. In the past there have been a lot of debate, and people from either side have been arguing to prove the superiority of their techniques. The fact of the matter is that both of these paradigms offer some unique advantages, and have limitations at the same time (a comparison between SMT and RBMT is given in [Thurmair, 2004]). This has forced the people from two different schools of thought to come close to each other resulting into a third approach: hybrid translation paradigm. This approach is under active development, and a number of experiments [Ahsan et al., 2010, Grishman and Kosaka, 1992, Carbonell et al., 1992] have been reported previously.

In one way of hybridization, one of the two approaches (i.e. linguistic and non-linguistic) is used as a principle translation technique, while the other is used to overcome the shortcomings of the main approach. An example is the use of linguistic rules and full-form lexicons in the statistical machine translation (SMT) systems. In this example, SMT is the principle translation technique, while linguistics rules and full-from lexicons are used to solve the data sparse issue.

Grammatical Framework (GF) [Ranta, 2004] is a grammar formalism tool and is suitable to build rule-based natural language processing (NLP) ap-

plications. GF provides a library of a set of parallel resource grammars [Ranta, 2009b] to assist its developers. GF and its resource grammar library were not originally designed for open-domain machine translation, but in the recent past the coverage of its resource grammars and its processing speed has grown to a level, where one can think about open-domain machine translation. However, for that, there are a number of challenges in-front including syntactic and lexical disambiguation. GF uses Parallel Multiple Context Free Grammars (PMCFG) [Seki et al., 1991, Ljunglöf, 2004] for parsing and suffers with syntactic ambiguity – one of the known issues with Context Free Grammars (CFG). Since natural languages are ambiguous, it is not surprising that for a sentence of length, say 20 tokens, the parser can return hundreds and even thousands of possible parse trees. How to disambiguate these trees (i.e. how to find the best possible syntactic analysis of a given sentence under a given CFG?). The usual way is to use statistics. This means, for open-domain machine translation in GF, one needs to think in terms of a hybrid approach, where the grammars can be used for parsing, and statistics can be used for disambiguation. Recently, the GF parser was extended to do syntactic disambiguation using a statistical model built from the Penn Treebank data [Angelov, 2011]. In this chapter, we describe our recent experiments for arbitrary text translation in GF, and also give directions for future work.

8.2 Our Recent Experiments

In GF, the translation works in a way analogous to compilers. The source language parser returns an abstract syntax tree which is then linearized to a target language. This means that there are two major translation phases: *parsing* and *linearization*. We did two rounds of experiments to discover issues related to each of the two phases, and to evaluate the systems. However, we also report a third round to deal with a more general issue of word-sense disambiguation (WSD). Let's start with round 1.

8.2.1 Round 1

In this round, we did not consider issues related to parsing, but concentrated only on the linearizarion phase. We took a set of around 2000 GF trees, which were produced by converting the Penn Treebank trees to GF format [Angelov, 2011]. The advantage of using the pre-parsed set was that these trees were guaranteed to be correct, and we did not need to take care of the parsing issues. We linearized this set of trees to German, Hindi and Urdu

Translation System	BLEU	WER	PER	TER
Google	0.31	0.56	0.43	0.53
GF	0.34	0.61	0.51	0.59

Table 8.1: The English-Hindi evaluation results for in-grammar sentences.

using their resource grammars and wide coverage lexicons. The evaluation results are given in the following subsections.

Evaluation

For the purpose of evaluation, a random set of 100 sentences was selected, and then two candidate translations were produced: one by using the GF translation system, and the second by Google. To avoid any biasing a random mixture of 100 translations (50 from GF, and the other 50 from Google) was given to informants, who were native speakers of the underlying language, for corrections. The corrected translations returned by the informants were used as a gold-standard. The two candidate translations were evaluated against the gold-standard using an automatic on-line evaluation tool: Asiya [Giménez and Màrquez, 2010]. The evaluation results are given in the following subsections for each of the language pairs:

English-Hindi

Table 8.1 shows different evaluation scores for the English-Hindi pair. We can see that our baseline translation system got better[1] BLEU scores than Google.

English-German

Table 8.2 shows evaluation scores for English-German pair. Google translations are better than GF, probably due to the rich amount of training data. Note that, also in GF system, the BLEU scores for German are better than the other languages. This has probably to do with the higher quality of the German grammar due to its long history of use in GF applications.

English-Urdu

For a different set of 30 random sentences – long sentences compared to the sentences used for English-Hindi, and English-German evaluations, the

[1]Of course, the comparison with Google translate is not fair, because we are evaluating the systems on in-grammar sentences using pre-parsed sentences.

Translation System	BLEU	WER	PER	TER
Google	0.63	0.25	0.19	0.23
GF	0.50	0.31	0.30	0.31

Table 8.2: The English-German evaluation results for in-grammar sentences.

Translation System	BLEU	WER	PER	TER
Google	0.22	0.71	0.49	0.65
GF	0.34	0.57	0.37	0.51

Table 8.3: The English-Urdu evaluation results for in-grammar sentences.

evaluation results for the English-Urdu pair are given in Table 8.3. The GF system is better than the Google system. The Goolge translations for Urdu are not very impressive, probability because of the unavailability of huge amounts of machine readable parallel data for English-Urdu pair.

Summary of Round 1

The BLEU scores in this round were very encouraging, not only competitive but better than Google translate for English-Hindi, and English-Urdu pairs. This suggests that we have an arbitrary text translator 'almost for free' from already existing resource grammars. But, let's first consider the second round of experiments before making such claims.

8.2.2 Round 2

Figure 8.1. shows the translation pipeline for this round. The architecture is hybrid, and it uses the Resource Grammar Library (RGL) of GF as the syntax and semantics component, a statistical model built from the Penn Treebank data for parse-tree disambiguation, and wide-coverage GF dictionaries as lexical components. Internal GF resources (e.g. resource grammars and dictionaries) are shown in rectangles while the external component (i.e. PennTreebank) is shown in double-stroked rectangle.

The input is parsed using an extended version of the English resource grammar (i.e. ParseEng in Figure 8.1) and a comprehensive English dictionary (i.e. DictEng in Figure 8.1). In cases where the input is syntactically ambiguous the parser returns more than one parse-trees. These trees are disambiguated using the statistical model. Finally, the best tree is linearized to a target language using an extended version of the target language resource grammar (i.e. TLPG in Figure 8.1) and its lexicon.

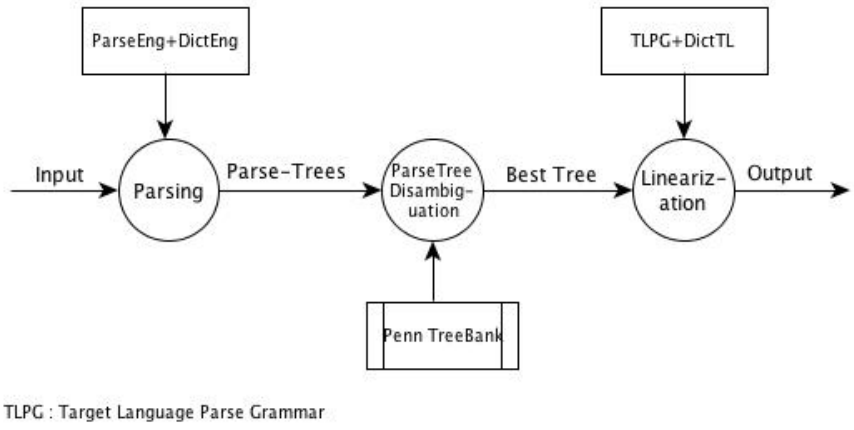

Figure 8.1: The translation pipeline.

There were a couple of issues in this phase including parsing speed and robustness. Let's first see how we deal with them for our current experiments, which is followed by the evaluation results.

- **Parsing Speed:** The parser sometimes takes very long time for long sentences, so we restricted our experiments to the sentences with length <=15 tokens.
- **Robustness:** The parser supports limited robustness and returns meta variables (i.e. '?') for the unknown tokens. In a parse tree these meta variables can appear both at the leaf nodes (i.e. lexical entries) or at the higher positions (i.e. function names). The GF linearizer, however, does not fully support partial parse-trees. The linearizer returns a linearization only if a tree has meta variables at the leaf node positions, but not if a tree has meta variables at a function name position. For these experiments, we considered only successfully linearized trees.

Evaluation

We took a set of 100 sentences (each with a length <=15 tokens), parsed it and linearized it to Urdu. 60 sentences out of these 100 sentences were successfully linearized (as explained in the previous section, the trees with meta variables at the position of function names are not supported). Table 8.4 gives our first results (here we give only the BLEU scores). If we compare the base-line BLEU scores (i.e. r1 in the Table 8.4) with the BLEU scores

Rounds	GF	Google
r1:Base Line	0.0232	0.1675
r2:After Lexical Filling	0.0339	0.1665
r3:After Construct Filling	0.0344	0.1665
r4:After Removing Meta Variables	0.0342	0.1665
r5:After Symbols Scripts Changed	0.0424	0.1665
r6:After Better Lexical Choices	0.0574	0.1665

Table 8.4: The English-Urdu evaluation results

of round 1, we can see a considerable difference. Round 1 suggests that GF system is better than Google for some languages, but the base-line scores in round 2 suggest it the other way around. Few initial observations and their effect on the BLEU scores are described below:

- **Missing Lexical Entries:** Consider the following sentence and its translation produced by the GF system.

  ```
  Source: the savagery of the attack has shocked the
     government and observers
  Translation: چھاپا کا savagery عمل داری اور مبصر صد چکا ہے
  ```

  The words that do not exist in the target language lexicon are preserved in the output e.g. 'savagery' in the above sentence. This have an effect on BLEU scores. Our scores improved from 0.023 to 0.033 (see r2 in the above table) after enriching the target language lexicon with the missing words.

- **Missing Functions:** If the target language grammar is missing some constructions can effect on the BLEU scores. r3 in the above table shows that the scores improved from 0.033 to 0.034 after enriching the target language grammar with the missing functions.

- **Meta Variables:** The GF system output might have meta variables e.g consider the following example:

  ```
  Source: pakistani squad left for West Indies via Dubai and
      London on Saturday
  Translation:
     دستہ ویسٹ انڈیز کیلیے دوبئی اور لندن کے ذریعے ہفتے پر رخا ?5
  ```

  In this example the word 'pakistani' was not parsed and the linearizer returned a meta variable '?5'. We observed a decline in the BLEU scores

(see r4 in the above table) after removing such meta variables from the output (probably because the number of word-count difference).

- **Symbols Script:** In the GF system, the proper names are parsed as special symbols and are not translated. However, for Urdu at least their script should be changed. After changing the script we observed an improvement from 0.034 to 0.042 (r5 in the above table)

- **Lexical Choices:** Our GF Urdu lexicon was compiled from different sources and was known to have low quality and sometimes wrong one-to-one word mappings. For example in the sentence:

  ```
  Source: pakistani squad left for West Indies via Dubai and
      London on Saturday
  Translation:
      5? دستہ ویسٹ انڈیز کیلیے دوبئی اور لندن کے ذریعے ہفتے پر رخا
  ```

  The word 'leave' has been wrongly mapped to رخصت instead of رخصت ہونا in the Urdu dictionary. A manual correction of such entries improved the scores from 0.042 to 0.057.

From the above given detailed analysis and corrections, we observed an overall improvement from 0.023 to 0.057. But, we can see that there is still a considerable difference between round 1 BLEU scores and the improved round 2 BLEU scores. We did some further analysis which lead us to the following further observations.

Further Observations

- **Word Sense Disambiguation:** Manual inspection of the results revealed that another reason for these low BLEU scores is word sense ambiguity. For example in the sentence:

  ```
  you are familiar with triangles and many of their properties
  from your earlier classes
  ```

  The noun 'property' was translated to جائیداد, jai:da:d – in the sense of 'belongings' rather than 'characteristics'. Currently, GF system does not support any Word Sense Disambiguation (WSD). However, in our in-progress work , we have added some support for WSD. Details are given in the next round.

- **Syntactic Transformations:** Another issue is syntactic transformations. In an inter-lingual translation approach, when the source and target languages do not express the common meaning in the same syntactic way, structural changes are needed. As an example, consider the sentence "I have few books". The GF abstract representation for this sentence is given in Figure 8.2. If we linearise this tree to Urdu, it will

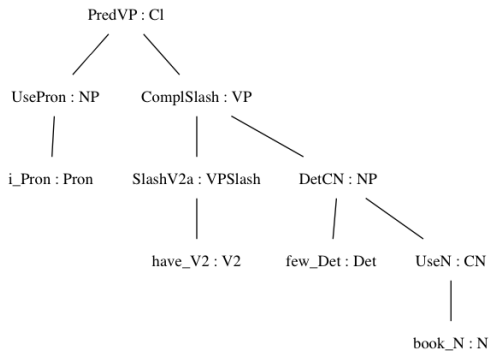

Figure 8.2: Abstract representation

produce the following translation:

میں چند کتابیں رکھتا ہوں *mẽ cand kita:bẽ rak^{h}ta: hõ*

However, this is not the preferred translation. The preferred one is

میرے پاس چند کتابیں ہیں *mere: pa:s cand kita:bẽ hẽ*

"in my possession are few books", which needs a structural change before linearisation. For high quality translations these structural changes are important. In the current experiment, we do not perform these changes, but this is an important direction for future work.

- **Choice of Prepositions:** Another major issue is the choice of a proposition. For example consider the following two sentence segments:

```
Use of mobile phone by students
Taken to the hospital by ambulance
```

If we translate them to Urdu, the preferred translation for the preposition 'by' in the first sentence segment is سے, while for the second sentence segment, the preferred translation is کے ذریعے. Our current translation system does not handle such issues.

- **Word-Order Preferences:** Another issue, perhaps more specific to Urdu/Hindi, is the word-order preferences in adverbial phrases. Urdu/Hindi tends to show different word-order preferences for different types of adverbs (e.g. time, place, manner etc.) in the final adverbial phrase. The followings are few observations:

  - Adverbs of time tend to come before adverbs of place. (e.g. here at 4-o-clock, "ca:r baje: yahã")
  - Adverbs of manner tend to come after the adverbs of time and place. (e.g. here at 4-o-clock happily, "ca:r baje: yahã khuxi: se:")
  - Several adverbs of the same kind are ordered bigger to smaller. (e.g. every evening at 4-o-clock, "ro:z xa:m ca:r baje:".

  Our current implementation does not distinguish between different types of adverbs, and hence does not reflect such preferences.

- **Stylistic and Idiomatic Issues:** There are many stylistic and idiomatic issues that need to be handled to produce a realistic translation in many cases. For example consider the phrase "an evening of music". The syntactic translation of this phrase to Urdu will produce "موسیقی کی ایک شام", which is perhaps acceptable but not realistic. The realistic translation will require a syntactic transformation from "an evening of music" to "music-gathering" before translation. Similarly, the translation of idiomatic expressions is another issue.

8.2.3 Round 3

In this round, we experimented to overcome the issue of word-sense ambiguity by integrating an external word-sense disambiguation tool into our translation pipeline. Let's first see the translation pipeline for this round.

System architecture

The translation pipeline for this round is similar to the translation pipeline of round 2 (see Figure 8.1) except that we have an extra component to deal with word sense disambiguation, and a different version of a target language lexicon is used. In Figure 8.3, IMS(It Makes Sense)[Zhong and Ng, 2010] is a word sense disambiguation tool and LinkdedDict refers to the full-sense lexicon, described in Chapter 5, of a target language.

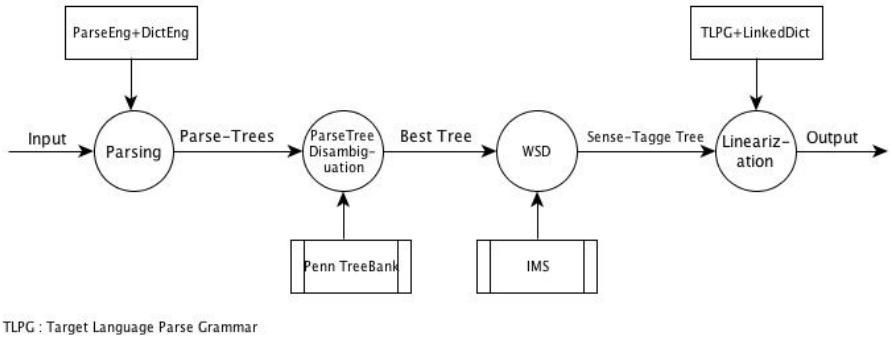

Figure 8.3: The translation pipeline with WSD.

Experimental Setup and Evaluation

Our experimental setup is as follows: We took English text, as source, and translated it to a target language (Hindi in these experiments) by passing it through the translation pipeline shown in Figure 8.3. To show the effects of word-sense disambiguation and for comparison purposes, we translated the same source twice: with and without word sense disambiguation using the translation pipelines given in Figure 8.3 and 8.1 respectively.

We compared both candidate translations to find if we have gained anything or not.

Next, we give a number of example sentence with comments to show that how the full-sense lexicons improved the quality of translations, and also give some examples where it worked the other way around.

Hindi

1. **Source** Mr Baris is a lawyer in New York .

   Without WSD `Mr Baris New York mẽ kānūn kā pañḍit hæ`

   With WSD `Mr Baris New York mẽ vakīl hæ`

   Word order `Mr Baris New York in lawyer is`

   Comments `kānūn kā pañḍit` is "expert/teacher in law", while `vakīl` means "lawyer".

2. **Source** we don't depend on pharmaceutical companies for our support

   Without WSD `ham auṣad`^{h}`īya sahyōgī par hamāre b`^{h}`araṇ pōṣaṇ ke liye nahī nirte hæ̃.`

With WSD ham auṣadʰīya kañpanī par hamāre nirvāh vyay ke liye nahī ūte hæ̃.

Word order We pharmaceutical companies on our subsistence expenditure for not ??? do

Comments sahyōgī means "company" in the sense of "colleagues", nirvāh vyay means "subsistence expenditure" , while bʰaraṇ pōṣaṇ means "weight bearing". The penultimate word in both versions is nonsense, and the lexicons need to be debugged.

3. **Source** you may recall that a triangle is also a polygon

   Without WSD tum "recall may" ho ki ṭrāyengl "also" bahubʰuj hæ

   With WSD tum smaraṇ kar sakte ho ki trikoṇ bʰī bahubʰuj hæ

   Word order You recall do can that triangle also polygon is

   Comments The version without WSD has several missing words. The WSD version of "recall" is not idiomatic, but understandable.

4. **Single words**
   (a) (human) right. Without WSD: dakṣiṇ right (not left), WSD: adʰikār.
   (b) security (of person). Without WSD: rṇpatr (as in commercial paper), WSD: surakṣā
   (c) property (in law) Without WSD: dʰan (means "wealth"), WSD: sampatti.
   (d) nationality (as citizenship). Without WSD: rāṣṭrīytā (could mean 'nationalism"), WSD: rāṣṭriktā.
   (e) comment. Without WSD: mat prakaṭ (announce opinion), WSD: ṭīkā ṭippaṇī (commentary)
   (f) pair (of socks). Without WSD: pati-patni (married couple), WSD: yugm

8.3 Future Directions

- One can try to improve the quality of lexicons, our observation is that the major bottleneck in getting high quality translations is the quality of lexicons.

- Better ways to do word-sense disambiguation need to be explored and one should try to integrate it within GF translation system, rather than doing it as a post-processing step.

- The current syntactic disambiguation in GF is based on context-free probabilities. For better syntactic disambiguation, one can try to use context-dependent probabilities and the structural preferences given in Collins Generative Models of Parsing [Collins, 1997]. As an experiment, we implemented and used the Collins first model, with some adjustments, as a post-processor to re-rank the trees returned by our current disambiguation model. We witnessed some improvements, but the results are not mature enough to be reported here.

- One can try to use statistics to deal with the issue of preposition choices. The probabilities of using a particular proposition with a particular word can be calculated from training data, and then can be used in the translation system.

- One needs to find a way to deal with the stylistic issues and the translation of idiomatic expressions to produce more realistic translations.

Specific to Hindi/Urdu

- One can try to categorize the adverbs into time, place, manner etc. and deal with the word-order preferences issue. One way to do it is to have an extra field in the linearization of the adverb category to store information about its type. Later, this information can be used to adjust different word-order preferences in the construction of the final adverbial phrase.

Appendix A

Hindi and Urdu Resource Grammars Implementation

A.1 Modular view of a Resource Grammar

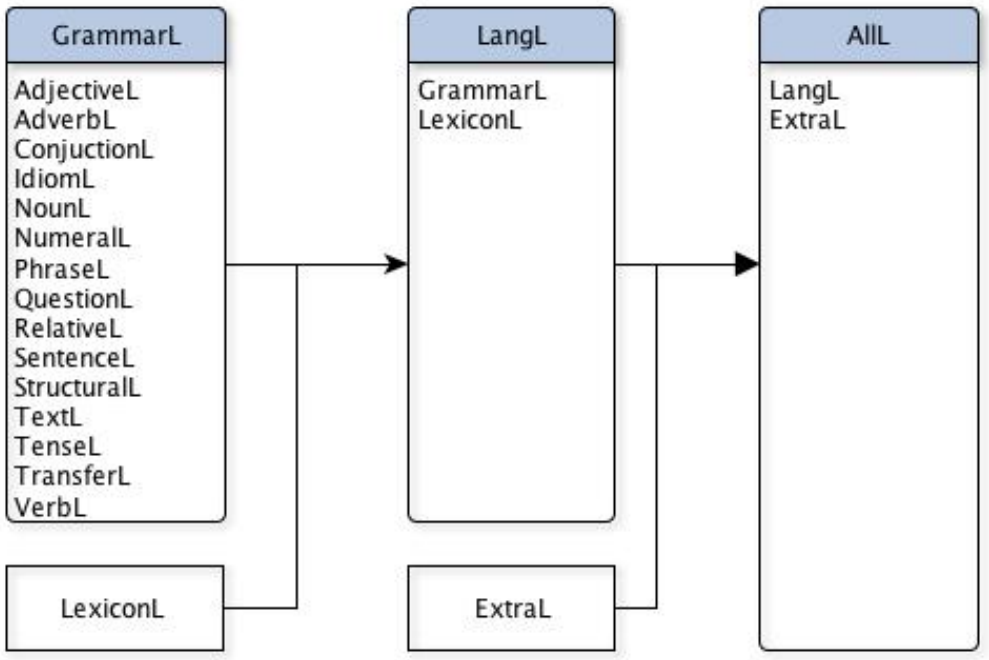

Figure A.1: Modular view of a Resource Grammar

GF provides a module system to support the division of labour. Figure A.1 shows different modules of a resource grammar. The three main modules are `GrammarL`, `LangL`, and `AllL`, where `L` stands for a three character ISO language code (e.g. 'Hin' for Hindi, 'Urd' for Urdu). A small description of each of these modules follows:

- The module `GrammarL` is further composed of 15 modules. Each of these 15 modules,except `StructuralL`, `TransferL` and `TenseL`, belongs to one of the GF resource grammar's syntactical categories, and contains construction rules specific to that particular category . The `StructuralL` module contains structural words (i.e. closed categories e.g. prepositions, determiners, quantifiers), `TransferL` defines experimental transfer rules (e.g 'active2passive'), and `TenseL` defines GF resource grammar library's tense system.
- The `GrammarL` module is combined with the lexical module `LexiconL` to build the `LangL` module. The `LexiconL` module has lexical entries

which are provided as a test lexicon with each resource grammar.

- `LangL` and `ExtraL` are then combined to produce a final module `AllL`. The `ExtraL` module contains language specific constructions, which are not accessible through the common resource grammar API, but can be accessed directly from this module.

A.2 Functor Style Implementation of Hindi and Urdu Resource Grammars

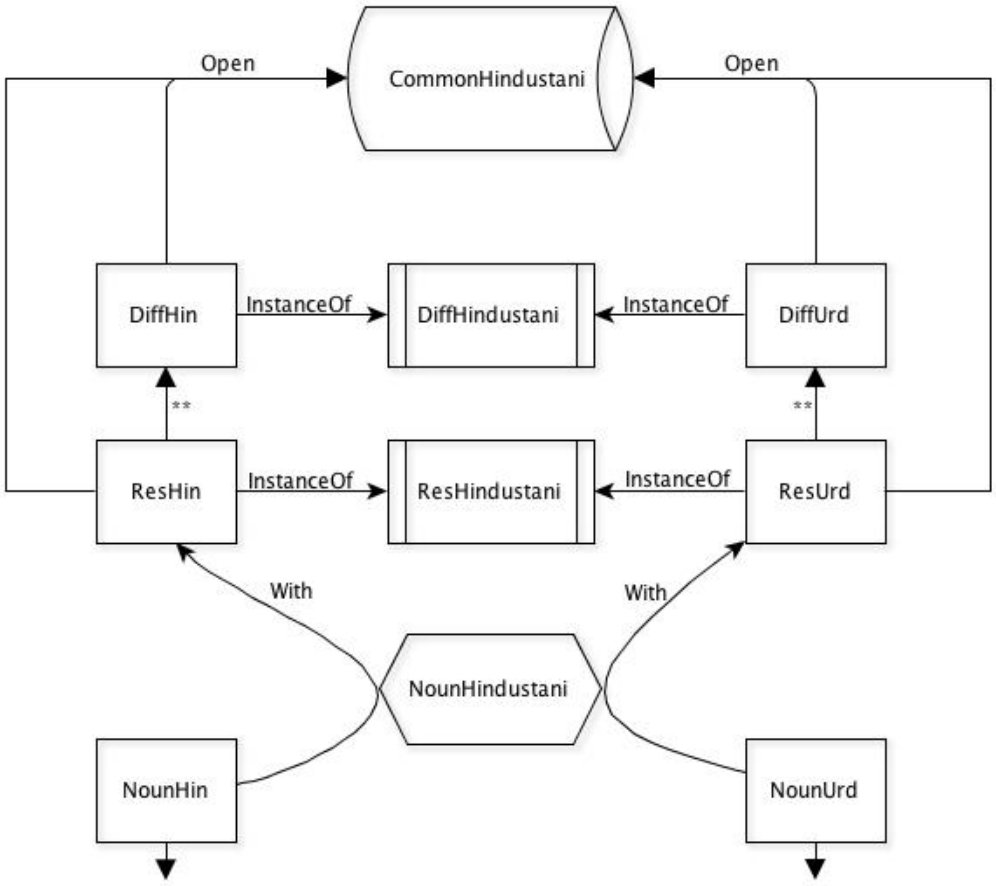

Figure A.2: Implementation of Hindi/Urdu Noun-Phrases through a Functor

Hindi and Urdu resource grammars have been implemented through a *functor*, which makes it possible to share the common code. The advantage of using *functors* is that they reduce the efforts required to implement a new language, and can also ease the task of bug fixing and maintenance. In the case of Hindi-Urdu, about 94% of the code was shared, and the implementation time reduced from months to days.

All syntactic modules were implemented through *functors* using the same principle, but here we briefly describe the development of Hindi/Urdu noun phrases only. Figure A.2 shows relationship between different modules diagrammatically, while their brief description follows.

NounHindustani–Functor

```
incomplete concrete NounHindustani of Noun =
  CatHindustani ** open CommonHindustani, ResHindustani, Prelude in {

  lin
    DetCN det cn = {
      s = \\c => detcn2NP det cn c det.n  ;
      a = agrP3 cn.g det.n
      } ;
.........
.........
}
```

The keyword `incomplete` (in the above code) declares `NounHindustani`[1] module to be a *functor*.

NounUrd–Concrete Module

```
concrete NounUrd of Noun = CatUrd ** NounHindustani with
  (ResHindustani = ResUrd) ;
```

This module defines the concrete grammar `NounUrd` of the abstract grammar `Noun` using the `ResUrd` instance of the `ResHindustani` interface.

NounHin–Concrete Module

```
concrete NounHin of Noun = CatHin ** NounHindustani with
  (ResHindustani = ResHin);
```

[1]We are using the term *Hindustani* to refer to the Hindi/Urdu base grammar

This module defines the concrete grammar `NounHin` of the abstract grammar `Noun` using the `ResHin` instance of the `ResHindustani` interface.

ResHindustani–Interface

```
interface ResHindustani =
    DiffHindustani ** open CommonHindustani, Prelude, Predef in {

param

    RAgr = RNoAg | RAg Agr ;
    RCase = RC Number Case ;

oper

    np2pronCase ppf npc a = case npc of {
       NPC c => ppf ! c;
       NPObj => ppf ! Obl ;
       NPErg =>case (fromAgr a).p of {
        (Pers3_Near|Pers3_Distant) => addErgative (ppf ! Obl) nE;
                  _      => addErgative (ppf ! Dir) nE
   }
      } ;
.........
.........
}
```

The `ResHindustani` interface extends the `DiffHindustnai` interface (note the extension operator **). This module contains definitions of all *parameters* and *operations* that are used to implement different syntactic rules of a resource grammar. For example, in the above code keyword `param` declares two parameters (i.e. `RAgr` and `RCase`) for the `agreement` and `case` features of a relative clause, while the keyword `oper` defines an operation (i.e. `np2pronCase`) for the construction of different noun-phrase cases. Note that the ergative case of a noun phrase is build using the function 'addErgative' which takes the ergative case marker, represented by the constant nE, and the direct or oblique case as arguments. These arguments are concatenated in the case of Urdu, but are glued together in the case of Hindi (see the DiffHin, and DiffUrd instances for more details). This shows how the differences between these two languages are implemented.

ResHin–Instance

```
instance ResHin of ResHindustani =
      DiffHin ** open CommonHindustani, Prelude, Predef in {} ;
```

The `ResHin` is an instance of the `ResHindustani` interface. It is an extension of the `DiffHin`, which is an instance of the `DifHindustani` interface.

ResUrd–Instance

```
instance ResUrd of ResHindustani =
     DiffUrd ** open CommonHindustani, Prelude, Predef in {} ;
```

The `ResUrd` is an instance of the `ResHindustani` interface. It is an extension of the `DiffUrd`, which is an instance of the `DifHindustani` interface.

DiffHindustani–Interface

```
interface DiffHindustani = open Prelude in {
 oper

  addErgative : Str -> Str -> Str ;
  copula : CTense -> Number -> UPerson -> Gender -> Str ;
  raha : Gender -> Number -> Str ;
  cka : Gender -> Number -> Str ;
  hw : UPerson -> Number -> Str ;
  hwa : Agr -> Str ;

  conjThat : Str ;
  kwd : Str ;
  ky : Str ;
  ka : Str ;
  agr : Str ;
  awr : Str ;
  jn : Str ;
  js : Str ;
  jw : Str ;
  kw : Str ;
  mt : Str ;
  nE : Str ;
```

```
  nh : Str ;
  sE : Str ;
  nE : Str ;
  hE : Str ;
  mein : Str ;
  na : Str ;
  nahen : Str ;
  xayad : Str ;
  kya : Str ;
}
```

The `DiffHindustani` interface contains the declarations of those *operations* whose implementation is different for Hindi and Urdu. These include some lexical items (e.g. ka, ky, awr etc.) that are used in the syntax part, and also the following syntax functions:

- `addErgative : Str -> Str -> Str`: The ergative-case constructor.
- `copula : CTense -> Number -> UPerson -> Gender -> Str`: The copula constructor.
- `raha : Gender -> Number -> Str`: The auxiliary verb 'raha:'.
- `cka : Gender -> Number -> Str`: The auxiliary verb 'cuka:'.
- `hw : UPerson -> Number -> Str`: The auxiliary verb 'ho'.
- `hwa : Agr -> Str`: The auxiliary verb 'hua:'.

DiffHin–Instance

The `DiffHin` instance defines the Hindi version of those *operations* defined in the `DiffHindustani` interface. The implementation of some of these operations follows.

```
instance DiffHin of DiffHindustani =
             open CommonHindustani, Prelude in {
oper
  addErgative s1 s2 = Prelude.glue s1 s2 ;

  raha : Gender -> Number -> Str = \g,n ->
          case <g,n> of {
            <Masc,Sg> => "रहा"; -- raha:
```

```
          <Masc,Pl> => "रहे"; -- rahe
          <Fem,_> => "रही" -- rahi:

          };

  cka : Gender -> Number -> Str = \g,n ->
          case <g,n> of {
          <Masc,Sg> => "चुका"; -- cuka:
          <Masc,Pl> => "चुके"; -- cuke
          <Fem,_> => "चुकी" -- cuki:

          };

  agr = "गर" ; -- agar
  awr = "और" ; -- aur
  ky = "की" ; -- ki:
  ka = "का" ; -- ka:
  jn = "जिन" ; -- jin
  js = "जिस" ; -- jis
  jw = "जो" ; -- jo
  kw = "को" ; -- ko
  mt = "मत" ; -- mat
  nE = "ने" ; -- ne
  nh = "न" ; -- na:
  sE = "से" ; -- se
  hE = "हे" ; -- hæ
  kwd = "खुद" ; -- xud
  nahen = "नहीं" ; -- nahĩ
  xayad = "शायद" ;-- ša:yad
  kya = "क्या" ; -- kya:
  mein = "में" ; -- mẽ

}
```

Note in the operation `addErgative`, the two string arguments are glued using the predefined 'glue' operator. This is different in Urdu where the *ergative* case marker is used as a separate word.

DiffUrd–Instance

Here we can see that the *addErgative* operation concatenates its two string arguments using the concatenation operator '++' as opposed to the 'glue' operator in `DiffHin`.

```
instance DiffUrd of DiffHindustani =
    open CommonHindustani, Prelude in {

oper
addErgative s1 s2 = s1 ++ s2 ;

raha : Gender -> Number -> Str = \g,n ->
          case <g,n> of {
            <Masc,Sg> => "رہا"; -- raha:
            <Masc,Pl> => "رہے"; -- rahe
            <Fem,_> => "رہی" -- rahi:

            };

  cka : Gender -> Number -> Str = \g,n ->
           case <g,n> of {
            <Masc,Sg> => "چکا"; -- cuka:
            <Masc,Pl> => "چکے"; -- cuke
            <Fem,_> => "چکی" -- cuki:

            };

  agr = "اگر" ; -- agar
  awr = "اور" ; -- aur
  ky = "کی" ; -- ki:
  ka = "کا" ; -- ka:
  jn = "جن" ; -- jin
  js = "جس" ; -- jis
  jw = "جو" ; -- jo
  kw = "کو" ; -- ko
  mt = "مت" ; -- mat
  nE = "نے" ; -- ne
  nh = "نا" ; -- na:
  sE = "سے" ; -- se
  hE = "ہے" ; -- hæ
```

```
  kwd = "خود" ; -- xud
  nahen = "نہیں" ; -- nahĩ
  xayad = "شاید" ; -- ša:yad
  kya = "کیا" ; -- kya:
  mein = "میں" ; -- mẽ

}
```

CommonHindustani–Resource Module

The `CommonHindustani` module contains the definitions which are common to both Hindi and Urdu. The code given below defines the structure of a verb phrase (i.e. `VPH`) and a noun phrase (i.e. `NP`), and their presence in this module means that they are shared among Hindi and Urdu resource grammars.

```
resource CommonHindustani =
    ParamX ** open Prelude, Predef in {

 oper

  VPH = {
      s    : VPHForm => {fin, inf : Str} ;
      obj  : {s : Str ; a : Agr} ;
      subj : VType ;
      comp : Agr => Str;
      inf : Str;
      ad  : Str;
      embComp : Str ;
      prog : Bool ;
      cvp : Str ;
      } ;
   NP : Type = {s : NPCase => Str ; a : Agr} ;
..........
..........
}
```

Appendix B

Resource Grammar Library API

Introduction

This appendix lists important Resource Grammar Library API's lexical and syntactical constructions with examples in seven languages: English, Hindi, Nepali, Persian, Punjabi, Sindhi, and Urdu. These examples have been automatically generated using the corresponding resource grammars. We have used the same scripts that were used to produce the GF Resource Grammar Library Synopsis[1] document with suitable changes for a paper-based version.

Explanations

Category	**Explanation**	**Example**
A	one-place adjective	*warm*
A2	two-place adjective	*divisible*
AP	adjectival phrase	*very warm*
AdA	adjective-modifying adverb	*very*
AdN	numeral-modifying adverb	*more than*
AdV	adverb directly attached to verb	*always*
Adv	verb-phrase-modifying adverb	*in the house*
Ant	anteriority	simultaneous, anterior
CAdv	comparative adverb	*more*
CN	common noun (without determiner)	*red house*
Card	cardinal number	*seven*
Cl	declarative clause, with all tenses	*she looks at this*
Comp	complement of copula, such as AP	*very warm*
Conj	conjunction	*and*
Det	determiner phrase	*those seven*
Digits	cardinal or ordinal in digits	*1,000/1,000th*
IAdv	interrogative adverb	*why*
IComp	interrogative complement of copula	*where*
IDet	interrogative determiner	*how many*
IP	interrogative pronoun	*who*
Imp	imperative	*look at this*
Interj	interjection	*alas*
N	common noun	*house*
N2	relational noun	*son*
N3	three-place relational noun	*connection*
NP	noun phrase (subject or object)	*the red house*
Num	number determining element	*seven*
Numeral	cardinal or ordinal in words	*five/fifth*
Ord	ordinal number (used in Det)	*seventh*
PConj	phrase-beginning conjunction	*therefore*
PN	proper name	*Paris*
Phr	phrase in a text	*but be quiet please*
Pol	polarity	positive, negative
Predet	predeterminer (prefixed Quant)	*all*
Prep	preposition, or just case	*in*
Pron	personal pronoun	*she*
QCl	question clause, with all tenses	*why does she walk*
QS	question	*where did she live*
Quant	quantifier ('nucleus' of Det)	*this/these*
RCl	relative clause, with all tenses	*in which she lives*
RP	relative pronoun	*in which*

[1]The original synopsis document is available at http://www.grammaticalframework.org/lib/doc/synopsis.html

RS	relative	*in which she lived*
S	declarative sentence	*she lived here*
SC	embedded sentence or question	*that it rains*
Subj	subjunction	*if*
Temp	temporal and aspectual features	past anterior
Tense	tense	present, past, future
Text	text consisting of several phrases	*He is here. Why?*
Utt	sentence, question, word...	*be quiet*
V	one-place verb	*sleep*
V2	two-place verb	*love*
V2A	verb with NP and AP complement	*paint*
V2Q	verb with NP and Q complement	*ask*
V2S	verb with NP and S complement	*tell*
V2V	verb with NP and V complement	*cause*
V3	three-place verb	*show*
VA	adjective-complement verb	*look*
VP	verb phrase	*is very warm*
VPSlash	verb phrase missing complement	*give to John*
VQ	question-complement verb	*wonder*
VS	sentence-complement verb	*claim*
VV	verb-phrase-complement verb	*want*
Voc	vocative or "please"	*my darling*

Lexical Categories

A,A2,N,N3,N3,V,VA,VS,VQ,VV,V2,V2A,V2S,V2Q,V2V,V3 are lexical categories and their constructors are given in the lexical paradigms at the end of the appendix.

Syntax Rules and Structural Words

Source 1: `../src/api/Constructors.gf`
Source 2: `../src/abstract/Structural.gf`

AP - adjectival phrase

Function	Type	Example
`mkAP`	A -> AP	Eng: *warm* Hin: गरम Nep: तातो Pes: گرم Pnb: نتا Snd: گرم Urd: گرم
`mkAP`	A -> NP -> AP	Eng: *warmer than Paris* Hin: पैरिस से गरम Nep: पेरिस भन्दा तातो Pes: گرم تر از پاریس Pnb: پیرس توں نتا Snd: Urd: پیرس سے گرم

`mkAP`	AP -> VP -> AP	Eng: *she is ready to sleep* Hin: वह सोने को तैयार है Nep: उनी सुत्न तयार छिन् Pes: او آماده خوابیدن است Pnb: Snd: Urd: وہ سونے کو تیار ہے
`mkAP`	AdA -> A -> AP	Eng: *very old* Hin: बहुत बूढ़ा Nep: धेरै बुढो Pes: خیلی پیر Pnb: بہت بوڈا Snd: تمام پوڑھی Urd: بہت بوڑھا
`mkAP`	AdA -> AP -> AP	Eng: *very very old* Hin: बहुत बहुत बूढ़ा Nep: धेरै धेरै बुढो Pes: خیلی خیلی پیر Pnb: بہت بہت بوڈا Snd: تمام تمام پوڑھی Urd: بہت بہت بوڑھا
`mkAP`	Conj -> AP -> AP -> AP	Eng: *old or young* Hin: बूढ़ा या जवान Nep: बुढो अथवा जवान Pes: پیر یا جوان Pnb: بوڈا یا جوان Snd: پوڑھی یا جوان Urd: بوڑھا یا جوان
`mkAP`	Conj -> ListAP -> AP	Eng: *old , big and warm* Hin: बूढ़ा , बड़ा और गरम Nep: बुढो , ठुलो र तातो Pes: پیر بزرگ و گرم Pnb: بوڈا وڈا تے نتا Snd: پوڑھی وڏو ۽ گرم Urd: بوڑھا بڑا اور گرم
`mkAP`	Ord -> AP	Eng: *oldest* Hin: सब से बूढ़ा Nep: सबभन्दा बुढो Pes: پیر ترین Pnb: بوڈا Snd: پوڑھی Urd: سب سے بوڑھا
`mkAP`	CAdv -> AP -> NP -> AP	Eng: *as old as she* Hin: इतना बूढ़ा जितना वह Nep: उनी जत्तीकै बुढो Pes: به اندازه ی او پیر Pnb: ایناں بوڈا جناں او Snd: جیءن پوڑھی جھڙو هوء Urd: اتنا بوڑھا جتنا وہ

AdA - adjective-modifying adverb

Function	Type	Example
`almost_AdA`	AdA	Eng: *almost red* Hin: तक़रीबन लाल Nep: झण्डै रातो Pes: تقریباً قرمز Pnb: تقریبا لال Snd: گھٹو کری ڳاڑھو Urd: تقریبا لال
`quite_Adv`	AdA	Eng: *quite* Hin: काफ़ी Nep: एकदम Pes: کاملاً Pnb: کہاںلسہ Snd: ڇڏڻ Urd: خاموش
`too_AdA`	AdA	Eng: *too warm* Hin: बहुत गरम Nep: पनि तातो Pes: خیلی گرم Pnb: بہت تتا Snd: بيحد گرم Urd: بہت گرم
`very_AdA`	AdA	Eng: *very warm* Hin: बहुत गरम Nep: धेरै तातो Pes: خیلی گرم Pnb: بہت تتا Snd: تمام گرم Urd: بہت گرم

AdN - numeral-modifying adverb

Function	Type	Example
`almost_AdN`	AdN	Eng: *almost eight* Hin: तक़रीबन आठ Nep: झण्डै आठ Pes: تقریباً هشت Pnb: تقریبا اٹھ Snd: گھٹو کری اٺ Urd: تقریبا آٹھ
`at_least_AdN`	AdN	Eng: *at least eight* Hin: कम से कम आठ Nep: कमसेकम आठ Pes: حداقل هشت Pnb: کم توں کم اٹھ Snd: گھٽ مى گھٽ اٺ Urd: کم از کم آٹھ
`at_most_AdN`	AdN	Eng: *at most eight* Hin: ज़्यादा से ज़्यादा आठ Nep: बढीमा आठ Pes: حداکثر هشت Pnb: زیادہ توں زیادہ اٹھ Snd: گھٽي کان گھٽو اٺ Urd: زیادہ سے زیادہ آٹھ

`mkAdN`	CAdv -> AdN	Eng: *more than eight* Hin: आठ से ज़्यादा Nep: भन्दा बढी आठ Pes: بیشتر از هشت Pnb: نالوں ہور اٹھ Snd: سان گڏ وڌيڪ اٺ Urd: آٹھ سے زیادہ

AdV - adverb directly attached to verb

Function	Type	Example
`always_AdV`	AdV	Eng: *always* Hin: हमेशा Nep: सधैँ Pes: همیشه Pnb: ہمیشہ Snd: هميشه Urd: ہمیشہ

Adv - verb-phrase-modifying adverb

Function	Type	Example
`everywhere_Adv`	Adv	Eng: *everywhere* Hin: हर जगह Nep: जाता तते Pes: هر جا Pnb: ہر تھاں Snd: هر هنڌ Urd: ہر جگہ
`here7from_Adv`	Adv	Eng: *from here* Hin: यहाँ से Nep: यहाँ बाट Pes: اینجا Pnb: ایتھوں Snd: هيڏانهن Urd: یہاں سے
`here_Adv`	Adv	Eng: *here* Hin: यहाँ Nep: यहाँ Pes: اینجا Pnb: ایتھے Snd: هتی Urd: یہاں
`mkAdv`	A -> Adv	Eng: *warmly* Hin: गरम Nep: तातो गरी Pes: گرم Pnb: تتے Snd: گرم Urd: گرم
`mkAdv`	Prep -> NP -> Adv	Eng: *in the house* Hin: घर में Nep: घर मा Pes: در خانه Pnb: گھر وچ Snd: گھر ۾ Urd: گھر میں

`mkAdv`	Subj -> S -> Adv	Eng: *when she sleeps* Hin: कब वह सोती है Nep: कहिले उनी सुत्छिन् Pes: وقتی که او می خوابد Pnb: کدوں او سوندی اے Snd: جڏهن هوءَ سمهي ٿي Urd: جب وہ سوتی ہے
`mkAdv`	CAdv -> A -> NP -> Adv	Eng: *more warmly than he* Hin: उस से ज़्यादा गरम Nep: उ भन्दा बढी तातो Pes: گرم بیشتر او Pnb: اونوں ہور تتے Snd: هو کان وڌيڪ گرم Urd: اس سے زیادہ گرم
`mkAdv`	AdA -> Adv -> Adv	Eng: *very warmly* Hin: बहुत गरम Nep: धेरै तातो गरी Pes: خیلی گرم Pnb: بہت تتے Snd: تمام گرم Urd: بہت گرم
`mkAdv`	Conj -> Adv -> Adv -> Adv	Eng: *here and now* Hin: यहाँ और अब Nep: यहाँ र अहीले Pes: اینجا و حالا Pnb: ایتھے تے ہون Snd: هتي ۽ هاڻي Urd: یہاں اور اب
`mkAdv`	Conj -> ListAdv -> Adv	Eng: *with her , here and now* Hin: उस के साथ , यहाँ और अब Nep: उनी सँग , यहाँ र अहीले Pes: با او اینجا و حالا Pnb: اونوں دے نال ایتھے تے ہون Snd: هو سان هتي ۽ هاڻي Urd: اس کے ساتھ یہاں اور اب
`somewhere_Adv`	Adv	Eng: *somewhere* Hin: कहीं Nep: कहीं Pes: جایی Pnb: کتے Snd: ڪٿي Urd: کہیں پر
`there7from_Adv`	Adv	Eng: *from there* Hin: वहाँ से Nep: त्यहाँ बाट Pes: آنجا Pnb: اوتھوں Snd: هتان Urd: وہاں سے
`there7to_Adv`	Adv	Eng: *there* Hin: वहाँ पर Nep: त्यहाँ सम्म Pes: آنجا Pnb: اوتھے Snd: هتي Urd: وہاں پر

there_Adv	Adv	Eng: *there* Hin: वहाँ Nep: त्यहाँ Pes: آنجا Pnb: اوتھے Snd: اتی Urd: وہاں

Ant - anteriority

Function	Type	Example
anteriorAnt	Ant	Eng: *she has slept* Hin: वह सो चुकी है Nep: उनी सुतेकिछिन् Pes: او خوابیده است Pnb: او سو گیی اے Snd: هوءَ سمهيل اهي Urd: وہ سو چکی ہے
simultaneousAnt	Ant	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुल्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوءَ سمهي ٿي Urd: وہ سوتی ہے

CAdv - comparative adverb

Function	Type	Example
as_CAdv	CAdv	Eng: *as* Hin: इतना Nep: जत्तीकै Pes: به اندازه ی Pnb: ایناں Snd: جيءَن Urd: اتنا
less_CAdv	CAdv	Eng: *less* Hin: कम Nep: कम Pes: کمتر Pnb: کٹ Snd: گهٽ Urd: کم
more_CAdv	CAdv	Eng: *more* Hin: ज़्यादा Nep: बढी Pes: بیشتر Pnb: ہور Snd: وڌيڪ Urd: زیادہ

CN - common noun (without determiner)

Function	Type	Example
`mkCN`	N -> CN	Eng: *house* Hin: घर Nep: घर Pes: خانه Pnb: گھر Snd: گھر Urd: گھر
`mkCN`	N2 -> NP -> CN	Eng: *mother of the king* Hin: Nep: राजा को आमा Pes: مادر پادشاه Pnb: بادشاہ دی ماں Snd: Urd: بادشاہ کی ماں
`mkCN`	N3 -> NP -> NP -> CN	Eng: *distance from this city to Paris* Hin: पैरिस का इस शहर से फ़ासला Nep: पेरिस देखि यो शहर सम्म को दुरी Pes: فاصله از این شهر تا پاریس Pnb: Snd: Urd: پیرس کا اس شہر سے فاصلہ
`mkCN`	N3 -> CN	Eng: *distance* Hin: फ़ासला Nep: दुरी Pes: فاصله Pnb: Snd: Urd: فاصلہ
`mkCN`	A -> N -> CN	Eng: *big house* Hin: बड़ा घर Nep: ठुलो घर Pes: خانه ی بزرگ Pnb: وڈا گھر Snd: وڏو گھر Urd: بڑا گھر
`mkCN`	A -> CN -> CN	Eng: *big blue house* Hin: बड़ा नीला घर Nep: ठुलो निलो घर Pes: خانه ی آبی بزرگ Pnb: وڈا نیلا گھر Snd: وڏو نيرو گھر Urd: بڑا نیلا گھر
`mkCN`	AP -> CN -> CN	Eng: *very big blue house* Hin: बहुत बड़ा नीला घर Nep: धेरै ठुलो निलो घर Pes: خانه ی آبی خیلی بزرگ Pnb: بہت وڈا نیلا گھر Snd: تمام وڏو نيرو گھر Urd: بہت بڑا نیلا گھر
`mkCN`	N -> RS -> CN	Eng: *man whom she loves* Hin: आदमी जिस को वह प्यार करती है Nep: मान्छे जो लाई उनी माया गर्छिन् Pes: مردی که او دوست دارد Pnb: بندہ جن نوں او پیار کردی اے Snd: ماڻهو جنهن سان هوء عشق کری ٿی Urd: آدمی جس کو وہ پیار کرتی ہے

mkCN	CN -> RS -> CN	Eng: *old man whom she loves* Hin: बूढ़ा आदमी जिस को वह प्यार करती है Nep: बुढो मान्छे जो लाई उनी माया गर्छिन् Pes: مرد پیری که او دوست دارد Pnb: بوڈا بندہ جن نوں او پیار کردی اے Snd: پوڑھی ماڻھو جنھن سان ھو۶ عشق ڪري ٿي Urd: بوڑھا آدمی جس کو وہ پیار کرتی ہے
mkCN	N -> Adv -> CN	Eng: *house on the hill* Hin: पहाड़ पर घर Nep: घर पहाड मा Pes: خانه ی روی تپه Pnb: گھر پاڑی اتے Snd: گھر ٽڪري مٿان Urd: پہاڑی پر گھر
mkCN	CN -> Adv -> CN	Eng: *big house on the hill* Hin: पहाड़ पर बड़ा घर Nep: ठुलो घर पहाड मा Pes: خانه ی بزرگ روی تپه Pnb: وڈا گھر پاڑی اتے Snd: وڏو گھر ٽڪري مٿان Urd: پہاڑی پر بڑا گھر
mkCN	CN -> VP -> CN	Eng: *reason to sleep* Hin: सोने की वजह Nep: सुत्न लाइ कारण Pes: دلیل خوابیدن Pnb: Snd: سبب سمهڻ Urd: سونے کی وجہ
mkCN	CN -> SC -> CN	Eng: *reason to sleep* Hin: सोने की वजह Nep: सुत्न लाइ कारण Pes: دلیل خوابیدن Pnb: Snd: سبب سمهڻ Urd: سونے کی وجہ
mkCN	N -> NP -> CN	Eng: *king John* Hin: राजा जान Nep: राजा जोन Pes: پادشاه جان Pnb: بادشاہ جان Snd: بادشاھ جان Urd: بادشاہ جان
mkCN	CN -> NP -> CN	Eng: *old king John* Hin: बूढ़ा राजा जान Nep: बुढो राजा जोन Pes: پادشاه پیر جان Pnb: بوڈا بادشاہ جان Snd: پوڑھی بادشاھ جان Urd: بوڑھا بادشاہ جان

Card - cardinal number

Function	Type	Example
`mkCard`	Numeral -> Card	Eng: *seven* Hin: सात Nep: सात Pes: هفت Pnb: ست Snd: ست Urd: سات

Cl - declarative clause, with all tenses

Function	Type	Example
`genericCl`	VP -> Cl	Eng: *one sleeps* Hin: कोई सोता है Nep: कोही सुत्छ Pes: آدم می خوابد Pnb: کوی سوندا اے Snd: کوءَ سمهی ٿو Urd: کوی سوتا ہے
`mkCl`	NP -> V -> Cl	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुत्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوءَ سمهی ٿي Urd: وہ سوتی ہے
`mkCl`	NP -> V2 -> NP -> Cl	Eng: *she loves him* Hin: वह उस को प्यार करती है Nep: उनी उ लाई माया गछिन् Pes: او او را دوست دارد Pnb: او اونوں پیار کردی اے Snd: هوءَ هو سان عشق کري ٿي Urd: وہ اس کو پیار کرتی ہے
`mkCl`	NP -> VV -> VP -> Cl	Eng: *she wants to sleep* Hin: वह सोना चाहती है Nep: उनी सुत्न चाहन्छिन् Pes: او می خواهد بخوابد Pnb: او سونا چاندی اے Snd: هوءَ سمهڻ چاهي ٿي Urd: وہ سونا چاہتی ہے
`mkCl`	NP -> VS -> S -> Cl	Eng: *she says that I sleep* Hin: वह कहती है कि मैं सोता हूँ Nep: उनी भन्छिन्की म सुत्छु Pes: او می گوید که من می خوابم Pnb: او کہدی اے کہ میں سوندا واں Snd: هوءَ چوي ٿي ت مان سمهان ٿو Urd: وہ کہتی ہے کہ میں سوتا ہوں
`mkCl`	NP -> VQ -> QS -> Cl	Eng: *she wonders who sleeps* Hin: वह हैरान होती है कि कौन सोता है Nep: उनी अचम्म हुन्छिन्की को सुत्छन् Pes: Pnb: او حیران ہوندی اے کہ کون سوندا اے Snd: هوءَ هيران ٿيي ٿي ت ڪير سمهي ٿو Urd: وہ حیران ہوتی ہے کہ کون سوتا ہے

mkCl	NP -> V2A -> NP -> A -> Cl	Eng: *she paints it red* Hin: वह इस को लाल रंग करती है Nep: उनी यो लाई रातो रँग लागाउँछिन् Pes: او آن را قرمز رنگ می کند Pnb: او اینوں لال رنگ کردی اے Snd: هوءَ هن کی ڳاڑهو رنگ کری ٿی Urd: وہ اس کو لال رنگ کرتی ہے
mkCl	NP -> V2A -> NP -> AP -> Cl	Eng: *she paints it red* Hin: वह इस को लाल रंग करती है Nep: उनी यो लाई रातो रँग लागाउँछिन् Pes: او آن را قرمز رنگ می کند Pnb: او اینوں لال رنگ کردی اے Snd: هوءَ هن کی ڳاڑهو رنگ کری ٿی Urd: وہ اس کو لال رنگ کرتی ہے
mkCl	NP -> V2S -> NP -> S -> Cl	Eng: *she answers to him that we sleep* Hin: वह उस को जवाब देती है कि हम सोते हैं Nep: उनी उ लाई उत्तर दिन्छिन्की हामीहरु सुत्छौं Pes: او به او جواب می دهد که ما می خوابیم Pnb: او اونوں جواب دیندی اے کہ اسی سوندے واں Snd: هوءَ هو کی جواب ڏيی ٿی ت اسان سمهون ٿا Urd: وہ اس کو جواب دیتی ہے کہ ہم سوتے ہیں
mkCl	NP -> V2Q -> NP -> QS -> Cl	Eng: *she asks him who sleeps* Hin: वह उस से पूछती है कि कौन सोता है Nep: उनी उ सँग सोछ्छिन्की को सुत्छन् Pes: او از او می پرسد چه کسی می خوابد Pnb: او اونوں پچھدی اے کہ کون سوندا اے Snd: هوءَ هو کان پڇی ٿی ت کير سمهی ٿو Urd: وہ اس سے پوچھتی ہے کہ کون سوتا ہے
mkCl	NP -> V2V -> NP -> VP -> Cl	Eng: *she begs him to sleep* Hin: वह उस से सोने की भीख मांगती है Nep: उनी उ लाई सुत्न आग्रह गर्छिन् Pes: او از او خواهش می کند بخوابد Pnb: او اونوں سونا دی مانگدی اے Snd: Urd: وہ اس سے سونے کی التجا کرتی ہے
mkCl	NP -> A -> Cl	Eng: *she is old* Hin: वह बूढ़ी है Nep: उनी बुढी छिन् Pes: او پیر است Pnb: او بوڈی اے Snd: هوءَ پوڙهی اهی Urd: وہ بوڑھی ہے
mkCl	NP -> A -> NP -> Cl	Eng: *she is older than he* Hin: वह उस से बूढ़ी है Nep: उनी उ भन्दा बुढी छिन् Pes: او پیر تر از او است Pnb: او اورے توں بوڈی اے Snd: هوءَ هو کان پوڙهی اهی Urd: وہ اس سے بوڑھی ہے
mkCl	NP -> A2 -> NP -> Cl	Eng: *she is married to him* Hin: वह उस से शादी शुदा है Nep: उनी उ सँग विवाहित छिन् Pes: او او متأهل است Pnb: □□□□□□□□□□ Snd: Urd: وہ اس سے شادی شدہ ہے

`mkCl`	NP -> AP -> Cl	Eng: *she is very old* Hin: वह बहुत बूढ़ी है Nep: उनी धेरै बुढी छिन् Pes: او خیلی پیر است Pnb: او بہت بوڈی اے Snd: هوءَ تمام پوڙهی اهی Urd: وہ بہت بوڑھی ہے
`mkCl`	NP -> NP -> Cl	Eng: *she is the woman* Hin: वह औरत है Nep: उनी आईमाई हुन् Pes: او زن است Pnb: او زنانی اے Snd: هوءَ استری اهی Urd: وہ عورت ہے
`mkCl`	NP -> CN -> Cl	Eng: *she is a* Hin: वह बूढ़ी औरत है Nep: उनी बुढी आईमाई हुन् Pes: او زن پیر است Pnb: او بوڈی زنانی اے Snd: هوءَ پوڙهی استری اهی Urd: وہ بوڑھی عورت ہے
`mkCl`	NP -> Adv -> Cl	Eng: *she is here* Hin: वह यहाँ है Nep: उनी यहाँ छिन् Pes: او اینجا است Pnb: او ایتھے اے Snd: هوءَ هتی اهی Urd: وہ یہاں ہے
`mkCl`	NP -> VP -> Cl	Eng: *she always sleeps* Hin: वह हमेशा सोती है Nep: उनी सधैँ सुत्छिन् Pes: او همیشه می خوابد Pnb: او ہمیشہ سوندی اے Snd: هوءَ هميشه سمهي ٿي Urd: وہ ہمیشہ سوتی ہے
`mkCl`	NP -> Cl	Eng: *there are many houses* Hin: बहुत घर हैं Nep: त्यहाँ थुप्रै घरहरु छन् Pes: تعداد زیادی خانه هستند Pnb: اوتھے بہت زیادہ گھر نے Snd: هتی گھٹا گھر اهن Urd: بہت زیادہ گھر ہیں
`mkCl`	NP -> RS -> Cl	Eng: *it is she who sleeps* Hin: वह है जो सोती है Nep: उनी हुन् जो सुत्छिन् Pes: او است که می خوابد Pnb: او اے جیڑی سوندی اے Snd: هوءَ اهي جهڙي سمهي ٿي Urd: وہ ہے جو سوتی ہے
`mkCl`	Adv -> S -> Cl	Eng: *it is here that she sleeps* Hin: यहाँ वह सोती है Nep: यहाँ उनी सुत्छिन् Pes: اینجا او می خوابد Pnb: ایتھے او سوندی اے Snd: هتي هوءَ سمهي ٿي Urd: یہاں وہ سوتی ہے

`mkCl`	V -> Cl	Eng: *it rains* Hin: बारिश होता है Nep: बर्षा हुन्छ Pes: باران می آید Pnb: بارش ہوندا اے Snd: Urd: بارش ہوتا ہے
`mkCl`	VP -> Cl	Eng: *it is raining* Hin: बारिश हो रहा है Nep: बर्षा भदै छ Pes: دارد باران می آید Pnb: بارش ہوندا پیا اے Snd: Urd: بارش ہو رہا ہے
`mkCl`	SC -> VP -> Cl	Eng: *that she sleeps is good* Hin: कि वह सोती है अच्छा है Nep: की उनी सुत्छिन् राम्रो छ Pes: این که او می خوابد خوب است Pnb: کہ او سوندی اے اچھا اے Snd: ت هوءَ سمهي ٿي سڻو اهي Urd: کہ وہ سوتی ہے اچھا ہے

ClSlash

Function	Type	Example
`mkClSlash`	NP -> V2 -> ClSlash	Eng: *whom does she see* Hin: किस को वह देखती है Nep: कासलाई उनी हेर्छिन् Pes: چه کسی را او می بیند Pnb: کون او ویکھدی اے Snd: کير هوءَ ڏسي ٿي Urd: کس کو وہ دیکھتی ہے
`mkClSlash`	NP -> VV -> V2 -> ClSlash	Eng: *whom does she want to see* Hin: किस को वह देखना चाहती है Nep: कासलाई उनी हेर्न चाहन्छिन् Pes: چه کسی را او می خواهد ببیند Pnb: کون او ویکھنا چاندی اے Snd: کير هوءَ ڏسڻ چاهي ٿي Urd: کس کو وہ دیکھنا چاہتی ہے
`mkClSlash`	ClSlash -> Adv -> ClSlash	Eng: *whom does she see today* Hin: किस को आज वह देखती है Nep: कासलाई आज उनी हेर्छिन् Pes: چه کسی را امروز او می بیند Pnb: کون اج او ویکھدی اے Snd: کير اڄ هوءَ ڏسي ٿي Urd: کس کو آج وہ دیکھتی ہے
`mkClSlash`	NP -> VS -> SSlash -> ClSlash	Eng: *whom does she know that we hadn't seen* Hin: किस को वह जानती है कि हम नहीं देख चुके थे Nep: कासलाई उनी थाहा पाउँछिन्की हामीहरु हेरेकाथिएनँ Pes: چه کسی را او می داند که ما ندیده بودیم Pnb: کون او جاندی اے کہ اسی نیں ویکھ گیے ساں Snd: کير هوءَ جاڻي ٿي ت اسان ڏسون ن هوءَسين Urd: کس کو وہ جانتی ہے کہ ہم نہیں دیکھ چکے تھے

Comp - complement of copula, such as AP

Function	Type	Example
`mkComp`	AP -> Comp	Eng: *to be old* Hin: बूढ़ा Nep: बुढो Pes: پیر بودن Pnb: بوڈا Snd: پوڑھی Urd: بوڑھا
`mkComp`	NP -> Comp	Eng: *to be this man* Hin: यह आदमी Nep: यो मान्छे Pes: این مرد بودن Pnb: اے بندہ Snd: هی ماڻهو Urd: یہ آدمی
`mkComp`	Adv -> Comp	Eng: *to be here* Hin: यहाँ Nep: यहाँ Pes: اینجا بودن Pnb: ایتھے Snd: هتی Urd: یہاں

Conj - conjunction

Function	Type	Example
`and_Conj`	Conj	Eng: *here and now* Hin: यहाँ और अब Nep: यहाँ र अहीले Pes: اینجا و حالا Pnb: ایتھے تے ہون Snd: هتی ۽ هاڻی Urd: یہاں اور اب
`both7and_DConj`	Conj	Eng: *both here and there* Hin: दोनों यहाँ और वहाँ Nep: दुबै यहाँ र त्यहाँ Pes: هم اینجا و هم آنجا Pnb: دوویں ایتھے تے اوتھے Snd: ٻ‌ئی هتی ۽ اتی Urd: دونوں یہاں اور وہاں
`if_then_Conj`	Conj	Eng: *if here then there* Hin: □□_□□□□_□□□□ Nep: यदि यहाँ भने त्यहाँ Pes: اگر اینجا آنگاه آنجا Pnb: اگر ایتھے تے اوتھے Snd: جيڪڏهن هتی ت اتی Urd: اگر یہاں تو وہاں
`or_Conj`	Conj	Eng: *here or there* Hin: यहाँ या वहाँ Nep: यहाँ अथवा त्यहाँ Pes: اینجا یا آنجا Pnb: ایتھے یا اوتھے Snd: هتی یا اتی Urd: یہاں یا وہاں

Det - determiner phrase

Function	Type	Example
`a_Det`	Det	Eng: *a* Hin: घर Nep: घर Pes: یک خانه Pnb: گھر Snd: گھر Urd: گھر
`every_Det`	Det	Eng: *every woman* Hin: हर औरत Nep: सबै आईमाई Pes: هر زن Pnb: ہر زنانی Snd: هر هک استری Urd: ہر عورت
`few_Det`	Det	Eng: *few women* Hin: चंद औरतें Nep: अलिकती आईमाईहरु Pes: تعداد کمی زن Pnb: کچھ زنانیاں Snd: کجھ استريون Urd: چند عورتیں
`many_Det`	Det	Eng: *many houses* Hin: बहुत घर Nep: थुप्रै घरहरु Pes: تعداد زیادی خانه Pnb: بہت زیادہ گھر Snd: گھڻا گھر Urd: بہت زیادہ گھر
`mkDet`	Quant -> Det	Eng: *this* Hin: यह Nep: यो Pes: این Pnb: اے Snd: هی Urd: یہ
`mkDet`	Quant -> Card -> Det	Eng: *these five* Hin: यह पांच Nep: यी पाँच Pes: این پنج Pnb: اے پنج Snd: هی پنج Urd: یہ پانچ
`mkDet`	Quant -> Ord -> Det	Eng: *the fifth* Hin: पांचवाँ Nep: पाँचौं Pes: پنجم پنجمین Pnb: پنج واں Snd: پنجون Urd: پانچ واں
`mkDet`	Quant -> Num -> Det	Eng: *these* Hin: यह Nep: यी Pes: این Pnb: اے Snd: هی Urd: یہ

`mkDet`	Card -> Det	Eng: *five* Hin: पांच Nep: पाँच Pes: یک پنج Pnb: پنج Snd: پنج Urd: پانچ
`mkDet`	Pron -> Num -> Det	Eng: *my five* Hin: मेरा पांच Nep: मेरो पाँच Pes: من پنج Pnb: میرا پنج Snd: پنج Urd: میرا پانچ
`somePl_Det`	Det	Eng: *some women* Hin: कुछ औरतें Nep: केही आईमाईहरु Pes: چند زن Pnb: کچھ زنانیاں Snd: کجھ استريون Urd: کچھ عورتیں
`someSg_Det`	Det	Eng: *some wine* Hin: कुछ शराब Nep: कोही वाईन Pes: مقداری شراب Pnb: کچھ شراب Snd: کجھ شراب Urd: کچھ شراب
`that_Det`	Det	Eng: *that woman* Hin: वह औरत Nep: त्ये आईमाई Pes: آن زن Pnb: وہ زنانی Snd: جيکو استری Urd: وہ عورت
`thePl_Det`	Det	Eng: *the houses* Hin: घर Nep: घरहरु Pes: خانه Pnb: گھر Snd: گھر Urd: گھر
`theSg_Det`	Det	Eng: *the house* Hin: घर Nep: घर Pes: خانه Pnb: گھر Snd: گھر Urd: گھر
`the_Det`	Det	Eng: *the house* Hin: घर Nep: घर Pes: خانه Pnb: گھر Snd: گھر Urd: گھر

`these_Det`	Det	Eng: *these women* Hin: ये औरतें Nep: यी आईमाईहरु Pes: این زن Pnb: اے زنانیاں Snd: هی استريون Urd: یہ عورتیں
`this_Det`	Det	Eng: *this woman* Hin: यह औरत Nep: यो आईमाई Pes: این زن Pnb: اے زنانی Snd: هی استری Urd: یہ عورت
`those_Det`	Det	Eng: *those women* Hin: वे औरतें Nep: यिनीहरु आईमाईहरु Pes: آن زن Pnb: وہ زنانیاں Snd: جيڪو استريون Urd: وہ عورتیں

Digits - cardinal or ordinal in digits

Function	Type	Example
`mkDigits`	Dig -> Digits	Eng: *4* Hin: ४ Nep: ४ Pes: □ Pnb: ۴ Snd: ۴ Urd: ۴
`mkDigits`	Dig -> Digits -> Digits	Eng: *1 , 2 3 3 , 4 8 6* Hin: ६८४३३२१ Nep: १२३३४८६ Pes: □ □ □ □ □ □ □ Pnb: ۶۸۴۳۳۲۱ Snd: ۶۸۴۳۳۲۱ Urd: ۶۸۴۳۳۲۱

IAdv - interrogative adverb

Function	Type	Example
`how8much_IAdv`	IAdv	Eng: *how much* Hin: कितना Nep: कती Pes: چقدر Pnb: کینا Snd: کيترا Urd: کتنا
`how_IAdv`	IAdv	Eng: *how* Hin: कैसे Nep: कसरी Pes: چطور Pnb: کسراں Snd: ڪ۶ين Urd: کیسے

`mkIAdv`	Prep -> IP -> IAdv	Eng: *in which city* Hin: कौन से शहर में Nep: कुन शहर मा Pes: در کدام شهر Pnb: Snd: Urd: کون سے شہر میں
`mkIAdv`	IAdv -> Adv -> IAdv	Eng: *where in Paris* Hin: कहाँ पैरिस में Nep: कहाँ पेरिस मा Pes: در پاریس کجا Pnb: کتھے پیرس وچ Snd: Urd: کہاں پیرس میں
`when_IAdv`	IAdv	Eng: *when* Hin: कब Nep: कहिले Pes: کی Pnb: کدوں Snd: ڪڏهن Urd: کب
`where_IAdv`	IAdv	Eng: *where* Hin: कहाँ Nep: कहाँ Pes: کجا Pnb: کتھے Snd: ڪٿي Urd: کہاں
`why_IAdv`	IAdv	Eng: *why* Hin: क्यों Nep: किन Pes: چرا Pnb: کیوں Snd: ڇو Urd: کیوں

IDet - interrogative determiner

Function	Type	Example
`how8many_IDet`	IDet	Eng: *how many houses* Hin: कितने घर Nep: कती वटा घरहरु Pes: چند خانه Pnb: کینے گھر Snd: ڪيترا گھر Urd: کتنے گھر
`mkIDet`	IQuant -> Num -> IDet	Eng: *which houses* Hin: कौन से घर Nep: कुन घरहरु Pes: کدام خانه Pnb: کیڑے گھر Snd: جيڪا گھر Urd: کون سے گھر

`mkIDet`	IQuant -> IDet	Eng: *which house* Hin: कौन सा घर Nep: कुन घर Pes: کدام خانه Pnb: کیڑا گھر Snd: جيڪو گھر Urd: کون سا گھر
`which_IDet`	IDet	Eng: *which house* Hin: कौन सा घर Nep: कुन घर Pes: کدام خانه Pnb: کیڑا گھر Snd: جيڪو گھر Urd: کون سا گھر

IP - interrogative pronoun

Function	Type	Example
`mkIP`	IDet -> CN -> IP	Eng: *which five big cities* Hin: कौन से पांच बड़े शहर Nep: कुन पाँच ठुला शहरहरु Pes: کدام پنج شهر بزرگ Pnb: کیڑے پنج وڈے شہر Snd: Urd: کون سے پانچ بڑے شہر
`mkIP`	IDet -> N -> IP	Eng: *which five cities* Hin: कौन से पांच शहर Nep: कुन पाँच शहरहरु Pes: کدام پنج شهر Pnb: کیڑے پنج شہر Snd: Urd: کون سے پانچ شہر
`mkIP`	IDet -> IP	Eng: *which five* Hin: कौन से पांच Nep: कुन पाँच Pes: کدام پنج Pnb: کیڑے پنج Snd: جيڪا پنج Urd: کون سے پانچ
`mkIP`	IQuant -> CN -> IP	Eng: *which big city* Hin: कौन सा बड़ा शहर Nep: कुन ठुलो शहर Pes: کدام شهر بزرگ Pnb: کیڑا وڈا شہر Snd: Urd: کون سا بڑا شہر
`mkIP`	IQuant -> Num -> CN -> IP	Eng: *which five big cities* Hin: कौन से पांच बड़े शहर Nep: कुन पाँच ठुला शहरहरु Pes: کدام پنج شهر بزرگ Pnb: کیڑے پنج وڈے شہر Snd: Urd: کون سے پانچ بڑے شہر

`mkIP`	IP -> Adv -> IP	Eng: *who in Paris* Hin: पैरिस में कौन Nep: पेरिस मा को Pes: چه کسی در پاریس Pnb: پیرس وچ کون Snd: Urd: پیرس میں کون
`whatPl_IP`	IP	Eng: *what* Hin: क्या Nep: के Pes: چه چیزهایی Pnb: Snd: ڇا Urd: کیا
`what_IP`	IP	Eng: *what* Hin: क्या Nep: के Pes: چه چیزی Pnb: کیا Snd: ڇا Urd: کیا
`whoSg_IP`	IP	Eng: *who* Hin: कौन Nep: को Pes: چه کسی Pnb: کون Snd: کير Urd: کون
`who_IP`	IP	Eng: *who* Hin: कौन Nep: को Pes: چه کسی Pnb: کون Snd: کير Urd: کون

IQuant

Function	Type	Example
`which_IQuant`	IQuant	Eng: *which house* Hin: कौन सा घर Nep: कुन घर Pes: کدام خانه Pnb: کیڑا گھر Snd: جيکو گھر Urd: کون سا گھر

Imp - imperative

Function	Type	Example
`mkImp`	V -> Imp	Eng: *come* Hin: आना Nep: आउ होउ Pes: بيا Pnb: اَنا Snd: Urd: اَنا

`mkImp`	V2 -> NP -> Imp	Eng: *buy it* Hin: इस को ख़रीदना Nep: यो किन् होउ Pes: آن را بخر Pnb: اینوں خریدنا Snd: ڏيد کر Urd: اس کو خریدنا

ImpForm

Function	Type	Example
`pluralImpForm`	ImpForm	Eng: *be men* Hin: आदमी Nep: मान्छेहरु हुऔ Pes: مردان Pnb: بندے Snd: مائھو Urd: آدمی
`politeImpForm`	ImpForm	Eng: *be a* Hin: आदमी Nep: मान्छे होउ Pes: مرد Pnb: بندہ Snd: مائھو Urd: آدمی
`singularImpForm`	ImpForm	Eng: *be a* Hin: आदमी Nep: मान्छे होउ Pes: مرد Pnb: بندہ Snd: مائھو Urd: آدمی

NP - noun phrase (subject or object)

Function	Type	Example
`everybody_NP`	NP	Eng: *everybody* Hin: हर कोई Nep: सबौ जाना Pes: Pnb: ہر کوی Snd: Urd: ہر کوی
`everything_NP`	NP	Eng: *everything* Hin: हर चीज़ Nep: हारेक कुरा Pes: Pnb: ہر شے Snd: Urd: ہر چیز
`it_NP`	NP	Eng: *it* Hin: यह Nep: यो Pes: آن Pnb: اے Snd: اھا Urd: یہ

mkNP	Quant -> N -> NP	Eng: *this man* Hin: यह आदमी Nep: यो मान्छे Pes: این مرد Pnb: اے بندہ Snd: هی ماڻهو Urd: یہ آدمی
mkNP	Quant -> CN -> NP	Eng: *this old man* Hin: यह बूढ़ा आदमी Nep: यो बुढो मान्छे Pes: این مرد پیر Pnb: اے بوڈا بندہ Snd: هی پوڙهی ماڻهو Urd: یہ بوڑھا آدمی
mkNP	Quant -> Num -> CN -> NP	Eng: *these five old men* Hin: ये पांच बूढ़े आदमी Nep: यी पाँच बुढा मान्छेहरु Pes: این پنج مرد پیر Pnb: اے پنج بوڈے بندے Snd: هی پنج پوڑها ماڻهو Urd: یہ پانچ بوڑھے آدمی
mkNP	Det -> CN -> NP	Eng: *the five old men* Hin: पांच बूढ़े आदमी Nep: पाँच बुढा मान्छेहरु Pes: پنج مرد پیر Pnb: پنج بوڈے بندے Snd: پنج پوڑها ماڻهو Urd: پانچ بوڑھے آدمی
mkNP	Det -> N -> NP	Eng: *the five men* Hin: पांच आदमी Nep: पाँच मान्छेहरु Pes: پنج مرد Pnb: پنج بندے Snd: پنج ماڻهو Urd: پانچ آدمی
mkNP	Digits -> CN -> NP	Eng: *5 1 old men* Hin: ५१ बूढ़े आदमी Nep: ५१ बुढा मान्छेहरु Pes: ۵۱ مرد پیر Pnb: ۵۱ بوڈے بندے Snd: ۵۱ پوڑها ماڻهو Urd: ۵۱ بوڑھے آدمی
mkNP	Digits -> N -> NP	Eng: *5 1 men* Hin: ५१ आदमी Nep: ५१ मान्छेहरु Pes: ۵۱ مرد Pnb: ۵۱ بندے Snd: ۵۱ ماڻهو Urd: ۵۱ آدمی
mkNP	Pron -> N -> NP	Eng: *my man* Hin: मेरा आदमी Nep: मेरो मान्छे Pes: مرد من Pnb: میرا بندہ Snd: ماڻهو Urd: میرا آدمی

mkNP	PN -> NP	Eng: *Paris* Hin: पैरिस Nep: पेरिस Pes: پاریس Pnb: پیرس Snd: □□□□□□□□□ Urd: پیرس
mkNP	Pron -> NP	Eng: *we* Hin: हम Nep: हामीहरु Pes: ما Pnb: اسی Snd: اسان Urd: ہم
mkNP	Quant -> Num -> NP	Eng: *these five* Hin: यह पांच Nep: यी पाँच Pes: این پنج Pnb: اے پنج Snd: هی پنج Urd: یہ پانچ
mkNP	CN -> NP	Eng: *old beer* Hin: बूढ़ा बियर Nep: बुढो बियर Pes: اَبجوی پیر Pnb: بوڈی شراب Snd: پوڑهی شراب Urd: بوڑھا بییر
mkNP	N -> NP	Eng: *beer* Hin: बियर Nep: बियर Pes: اَبجو Pnb: شراب Snd: شراب Urd: بییر
mkNP	Predet -> NP -> NP	Eng: *only this woman* Hin: सिर्फ़ यह औरत Nep: मात्र यो आईमाई Pes: فقط این زن Pnb: صرف اے زنانی Snd: صرف هی استری Urd: صرف یہ عورت
mkNP	NP -> V2 -> NP	Eng: *the man seen* Hin: देखा हुआ आदमी Nep: मान्छे हेर् एको Pes: مرد دیده شده Pnb: ویکھیا بندہ Snd: ڏسيل ماڻهو Urd: دیکھا ہوا آدمی
mkNP	NP -> Adv -> NP	Eng: *Paris today* Hin: आज पैरिस Nep: पेरिस आज Pes: پاریس امروز Pnb: پیرس اج Snd: Urd: آج پیرس

mkNP	NP -> RS -> NP	Eng: *John , who walks* Hin: जान , जो चलता है Nep: जोन , जो हिड्छ Pes: جان که راه می رود Pnb: جان جیڑا چلدا اے Snd: جان جھڑا هلی ٿو Urd: جان جو چلتا ہے
mkNP	Conj -> NP -> NP -> NP	Eng: *this woman or John* Hin: यह औरत या जान Nep: यो आईमाई अथवा जोन Pes: این زن یا جان Pnb: اے زنانی یا جان Snd: هی استری یا جان Urd: یہ عورت یا جان
mkNP	Conj -> ListNP -> NP	Eng: *this woman , John or I* Hin: यह औरत , जान या मैं Nep: यो आईमाई , जोन अथवा म Pes: این زن جان یا من Pnb: اے زنانی جان یا میں Snd: هی استری جان یا مان Urd: یہ عورت جان یا میں
that_NP	NP	Eng: *that* Hin: वह Nep: त्यो Pes: آن Pnb: وہ Snd: جيڪو Urd: وہ
these_NP	NP	Eng: *these* Hin: यह Nep: यी Pes: این Pnb: اے Snd: هی Urd: یہ
this_NP	NP	Eng: *this* Hin: यह Nep: यो Pes: این Pnb: اے Snd: هی Urd: یہ
those_NP	NP	Eng: *those* Hin: वह Nep: यिनीहरु Pes: آن Pnb: وہ Snd: جيڪو Urd: وہ

Num - number determining element

Function	Type	Example
`mkNum`	Digits -> Num	Eng: *2 1* Hin: १२ Nep: २१ Pes: Pnb: ۱۲ Snd: ۱۲ Urd: ۱۲
`mkNum`	Card -> Num	Eng: *almost five* Hin: तक़रीबन पांच Nep: झण्डै पाँच Pes: تقریباً پنج Pnb: تقریبا پنج Snd: گھٹو کری پنج Urd: تقریبا پانچ
`mkNum`	AdN -> Card -> Num	Eng: *almost five* Hin: तक़रीबन पांच Nep: झण्डै पाँच Pes: تقریباً پنج Pnb: تقریبا پنج Snd: گھٹو کری پنج Urd: تقریبا پانچ

Ord - ordinal number (used in Det)

Function	Type	Example
`mkOrd`	A -> Ord	Eng: *smallest* Hin: सब से छोटा Nep: सबभन्दा सानो Pes: کوچک ترین Pnb: چھوٹا Snd: ننڍو Urd: سب سے چھوٹا

PConj - phrase-beginning conjunction

Function	Type	Example
`but_PConj`	PConj	Eng: *but* Hin: लेकिन Nep: तर Pes: اما Pnb: مگر Snd: پر Urd: لیکن
`mkPConj`	Conj -> PConj	Eng: *and now* Hin: और अब Nep: र अहीले Pes: و حالا Pnb: تے ہون Snd: ۽ هائی Urd: اور اب

`otherwise_PConj`	PConj	Eng: *otherwise* Hin: नहीं तो Nep: अन्यथा Pes: درغیراین صورت Pnb: یا فیر Snd: ن ت پوءی Urd: یا پھر
`therefore_PConj`	PConj	Eng: *therefore* Hin: इस लिये Nep: अत: Pes: به همین دلیل Pnb: اس لی Snd: ان کری Urd: اس لیے

PN - proper name

Lexical category, constructors given in lexical paradigms.

Phr - phrase in a text

Function	Type	Example
`mkPhr`	S -> Phr	Eng: *she won't sleep* Hin: वह नहीं सोएगी Nep: उनी सुत्नेछैनन् Pes: او نخواهد خوابید Pnb: او نیں سوئے گی Snd: هوء سمهندی ن هوندی Urd: وہ نہیں سوئے گی
`mkPhr`	Cl -> Phr	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुत्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوء سمهی ٿی Urd: وہ سوتی ہے
`mkPhr`	QS -> Phr	Eng: *would she sleep* Hin: क्या वह सोएगी Nep: के उनी सुत्नेछिन् Pes: آیا او می خوابید Pnb: کی او شاید سوئے Snd: چا هوء شاید سمهندی Urd: کیا وہ سوئے گی

Pol - polarity

Function	Type	Example
`negativePol`	Pol	Eng: *she doesn't sleep* Hin: वह नहीं सोती है Nep: उनी सुत्दिन्न Pes: او نمی خوابد Pnb: او نیں سوندی اے Snd: هوء سمهی ن ٿی Urd: وہ نہیں سوتی ہے

positivePol	Pol	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुल्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوءَ سمهي ٿي Urd: وہ سوتی ہے

Predet - predeterminer (prefixed Quant)

Function	Type	Example
all_Predet	Predet	Eng: *all the men* Hin: तमाम आदमी Nep: सबै मान्छेहरु Pes: همه ی مرد Pnb: سارے بندے Snd: سڀ ماڻهو Urd: تمام آدمی
most_Predet	Predet	Eng: *most* Hin: सब से ज़्यादा Nep: ज्यादै Pes: اکثر Pnb: زیادہ تر Snd: سڀ کان گھٽو Urd: زیادہ تر
only_Predet	Predet	Eng: *only* Hin: सिर्फ़ Nep: मात्र Pes: فقط Pnb: صرف Snd: صرف Urd: صرف

Prep - preposition, or just case

Function	Type	Example
above_Prep	Prep	Eng: *above it* Hin: इस के ऊपर Nep: यो माथि Pes: بالای آن Pnb: اینوں اتے Snd: هن مٿي Urd: اس کے اوپر
after_Prep	Prep	Eng: *after it* Hin: इस के बाद Nep: यो पछि Pes: بعد از آن Pnb: اینوں توں بعد Snd: هن کان پوءِ Urd: اس کے بعد
before_Prep	Prep	Eng: *before it* Hin: इस से पहले Nep: यो अघि Pes: قبل از آن Pnb: اینوں پلے Snd: هن پيهرين Urd: اس سے پہلے

behind_Prep	Prep	Eng: *behind it* Hin: इस के पीछे Nep: यो पछि Pes: پشت آن Pnb: اینوں پیچھے Snd: هن پٺتی Urd: اس کے پیچھے
by8agent_Prep	Prep	Eng: *by it* Hin: इस से Nep: यो लाइ Pes: توسط آن Pnb: اینوں Snd: هن هٿان Urd: اس سے
by8means_Prep	Prep	Eng: *by it* Hin: इस पर Nep: यो ले Pes: با آن Pnb: اینوں Snd: هن کان Urd: اس پر
during_Prep	Prep	Eng: *during it* Hin: इस के दरमियान Nep: यो पर्यन्त Pes: در طول آن Pnb: اینوں دے وچ Snd: هن وچ ۾ Urd: اس کے درمیاں
except_Prep	Prep	Eng: *except it* Hin: इस के सिवाय Nep: यो बाहेक Pes: به جز آن Pnb: اینوں سوائے Snd: هن سواءِ Urd: اس کے سوائے
for_Prep	Prep	Eng: *for it* Hin: इस के लिये Nep: यो लागि Pes: برای آن Pnb: اینوں □دے واسط□□ Snd: هن لاءِ Urd: اس کیلیے
from_Prep	Prep	Eng: *from it* Hin: इस से Nep: यो बाट Pes: از آن Pnb: اینوں توں Snd: هن وٽان Urd: اس سے
in8front_Prep	Prep	Eng: *in front of it* Hin: इस के सामने Nep: यो सामु Pes: جلوی آن Pnb: اینوں دے سامنے Snd: هن جي سامهون Urd: اس کے سامنے

in_Prep	Prep	Eng: *in it* Hin: इस में Nep: यो मा Pes: در آن Pnb: اینوں وچ Snd: هن ۾ Urd: اس میں
on_Prep	Prep	Eng: *on it* Hin: इस पर Nep: यो मा Pes: روی آن Pnb: اینوں اتے Snd: هن مٿان Urd: اس پر
possess_Prep	Prep	Eng: *of it* Hin: इस का Nep: यो धारणा गर्नु Pes: آن Pnb: اینوں دا Snd: هن جو يا جي Urd: اس کا
through_Prep	Prep	Eng: *through it* Hin: इस में से Nep: यो मार्फत Pes: از طریق آن Pnb: اینوں وچوں Snd: هن منجهان Urd: اس میں سے
to_Prep	Prep	Eng: *to it* Hin: इस को Nep: यो सम्म Pes: به آن Pnb: اینوں نوں Snd: هن ڏانهن Urd: اس کو
under_Prep	Prep	Eng: *under it* Hin: इस के नीचे Nep: यो अन्तर्गत Pes: زیر آن Pnb: اینوں تھلے Snd: هن هيٺان Urd: اس کے نیچے
with_Prep	Prep	Eng: *with it* Hin: इस के साथ Nep: यो सँग Pes: با آن Pnb: اینوں دے نال Snd: هن سان Urd: اس کے ساتھ
without_Prep	Prep	Eng: *without it* Hin: Nep: यो विना Pes: بدون آن Pnb: اینوں توں بغیر Snd: هن کان سواءِ Urd: اس کے بغیر

Pron - personal pronoun

Function	Type	Example
`he_Pron`	Pron	Eng: *he* Hin: वह Nep: उ Pes: او Pnb: او Snd: هو Urd: وہ
`i_Pron`	Pron	Eng: *I* Hin: मैं Nep: म Pes: من Pnb: میں Snd: مان Urd: میں
`it_Pron`	Pron	Eng: *it* Hin: यह Nep: यो Pes: آن Pnb: اے Snd: اها Urd: یہ
`she_Pron`	Pron	Eng: *she* Hin: वह Nep: उनी Pes: او Pnb: او Snd: هوءَ Urd: وہ
`they_Pron`	Pron	Eng: *they* Hin: वे Nep: उनीहरु Pes: آن ها Pnb: او Snd: اهی Urd: وہ
`we_Pron`	Pron	Eng: *we* Hin: हम Nep: हामीहरु Pes: ما Pnb: اسی Snd: اسان Urd: ہم
`youPl_Pron`	Pron	Eng: *you* Hin: तुम Nep: तिमीहरु Pes: شما Pnb: تسی Snd: توهان Urd: تم
`youPol_Pron`	Pron	Eng: *you* Hin: आप Nep: तपाई Pes: شما Pnb: تسی Snd: توهان Urd: آپ

`youSg_Pron`	Pron	Eng: *you* Hin: तू Nep: तिमी Pes: تو Pnb: توں Snd: تون Urd: تو

Punct

Function	Type	Example
`exclMarkPunct`	Punct	Eng: *yes !* Hin: हाँ Nep: हजुर Pes: بله Pnb: ہاں Snd: ها Urd: ہاں
`fullStopPunct`	Punct	Eng: *yes .* Hin: हाँ Nep: हजुर Pes: بله Pnb: ہاں Snd: ها Urd: ہاں
`questMarkPunct`	Punct	Eng: *yes ?* Hin: हाँ Nep: हजुर Pes: بله Pnb: ہاں Snd: ها Urd: ہاں

QCl - question clause, with all tenses

Function	Type	Example
`mkQCl`	Cl -> QCl	Eng: *does she sleep* Hin: क्या वह सोती है Nep: के उनी सुत्छिन् Pes: آیا او می خوابد Pnb: کی او سوندی اے Snd: ڇا هوءَ سمهي ٿي Urd: کیا وہ سوتی ہے
`mkQCl`	IP -> VP -> QCl	Eng: *who always sleeps* Hin: कौन हमेशा सोता है Nep: को सधैँ सुत्छन् Pes: چه کسی همیشه می خوابد Pnb: کون ہمیشہ سوندا اے Snd: کير هميشه سمهي ٿو Urd: کون ہمیشہ سوتا ہے
`mkQCl`	IP -> V -> QCl	Eng: *who sleeps* Hin: कौन सोता है Nep: को सुत्छन् Pes: چه کسی می خوابد Pnb: کون سوندا اے Snd: کير سمهي ٿو Urd: کون سوتا ہے

`mkQCl`	IP -> V2 -> NP -> QCl	Eng: *who loves her* Hin: कौन उस को प्यार करता है Nep: को उनी लाई माया गर्छन् Pes: چه کسی او را دوست دارد Pnb: کون اونوں پیار کردا اے Snd: کير هو سان عشق کری ٿو Urd: کون اس کو پیار کرتا ہے
`mkQCl`	IP -> VV -> VP -> QCl	Eng: *who wants to sleep* Hin: कौन सोना चाहता है Nep: को सुत्न चाहन्छन् Pes: چه کسی می خواهد بخوابد Pnb: کون سونا چاندا اے Snd: کير سمهڻ چاهی ٿو Urd: کون سونا چاہتا ہے
`mkQCl`	IP -> VS -> S -> QCl	Eng: *who says that I sleep* Hin: कौन कहता है कि मैं सोता हूँ Nep: को भन्छन्की म सुत्छु Pes: چه کسی می گوید که من می خوابم Pnb: کون کہدا اے کہ میں سوندا واں Snd: کير چوی ٿو ت مان سمهان ٿو Urd: کون کہتا ہے کہ میں سوتا ہوں
`mkQCl`	IP -> VQ -> QS -> QCl	Eng: *who wonders who sleeps* Hin: कौन हैरान होता है कि कौन सोता है Nep: को अचम्म हुन्छन्की को सुत्छन् Pes: Pnb: کون حیران ہوندا اے کہ کون سوندا اے Snd: کير هيران ٿيی ٿو ت کير سمهی ٿو Urd: کون حیران ہوتا ہے کہ کون سوتا ہے
`mkQCl`	IP -> VA -> A -> QCl	Eng: *who becomes old* Hin: कौन बूढ़ा बनता है Nep: को बुढो हुन्छन् Pes: چه کسی پیر می شود Pnb: کون بوڈا بندا اے Snd: کير پوڙهی ٿيی ٿو Urd: کون بوڑھا بنتا ہے
`mkQCl`	IP -> V2A -> NP -> A -> QCl	Eng: *who paints it red* Hin: कौन इस को लाल रंग करता है Nep: को यो लाई रातो रँग लागाउँछन् Pes: چه کسی آن را قرمز رنگ می کند Pnb: کون اینوں لال رنگ کردا اے Snd: کير هن کی ڳاڙهو رنگ کری ٿو Urd: کون اس کو لال رنگ کرتا ہے
`mkQCl`	IP -> V2S -> NP -> S -> QCl	Eng: *who answers to him that we sleep* Hin: कौन उस को जवाब देता है कि हम सोते हैं Nep: को उ लाई उत्तर दिन्छन्की हामीहरु सुत्छौं Pes: چه کسی به او جواب می دهد که ما می خوابیم Pnb: کون اونوں جواب دیندا اے کہ اسی سوندے واں Snd: کير هو کی جواب ڏيی ٿو ت اسان سمهون ٿا Urd: کون اس کو جواب دیتا ہے کہ ہم سوتے ہیں
`mkQCl`	IP -> V2Q -> NP -> QS -> QCl	Eng: *who asks him who sleeps* Hin: कौन उस से पूछता है कि कौन सोता है Nep: को उ सँग सोध्छन्की को सुत्छन् Pes: چه کسی از او می پرسد چه کسی می خوابد Pnb: کون اونوں پچھدا اے کہ کون سوندا اے Snd: کير هو کان پڇی ٿو ت کير سمهی ٿو Urd: کون اس سے پوچھتا ہے کہ کون سوتا ہے

`mkQCl`	IP -> V2V -> NP -> VP -> QCl	Eng: *who begs him to sleep* Hin: कौन उस से सोने की भीख मांगता है Nep: को उ लाई सुत्न आग्रह गर्छन् Pes: چه کسی از او خواهش می کند بخوابد Pnb: کون اونوں سونا دی مانگدا اے Snd: Urd: کون اس سے سونے کی التجا کرتا ہے
`mkQCl`	IP -> A -> QCl	Eng: *who is old* Hin: कौन बूढ़ा है Nep: को बुढो छन् Pes: چه کسی پیر است Pnb: کون بوڈا اے Snd: کير پوڙهی اهی Urd: کون بوڑھا ہے
`mkQCl`	IP -> A -> NP -> QCl	Eng: *who is older than he* Hin: कौन उस से बूढ़ा है Nep: को उ भन्दा बुढो छन् Pes: چه کسی پیر تر از او است Pnb: کون اورے توں بوڈا اے Snd: کير هو کان پوڙهی اهی Urd: کون اس سے بوڑھا ہے
`mkQCl`	IP -> AP -> QCl	Eng: *who is very old* Hin: कौन बहुत बूढ़ा है Nep: को धेरै बुढो छन् Pes: چه کسی خیلی پیر است Pnb: کون بہت بوڈا اے Snd: کير تمام پوڙهی اهی Urd: کون بہت بوڑھا ہے
`mkQCl`	IP -> NP -> QCl	Eng: *who is the woman* Hin: कौन औरत है Nep: को आईमाई हुन् Pes: چه کسی زن است Pnb: کون زنانی اے Snd: کير استری اهی Urd: کون عورت ہے
`mkQCl`	IP -> Adv -> QCl	Eng: *who is here* Hin: कौन यहाँ है Nep: को यहाँ छन् Pes: چه کسی اینجا است Pnb: کون ایتھے اے Snd: کير هتی اهی Urd: کون یہاں ہے
`mkQCl`	IP -> ClSlash -> QCl	Eng: *whom does she love today* Hin: किस को आज वह प्यार करती है Nep: कासलाई आज उनी माया गर्छिन् Pes: چه کسی را امروز او دوست دارد Pnb: کون اج او پیار کردی اے Snd: کير اڄ هو۶ عشق کری ٿی Urd: کس کو آج وہ پیار کرتی ہے
`mkQCl`	IAdv -> Cl -> QCl	Eng: *why does she sleep* Hin: क्यों वह सोती है Nep: किन उनी सुत्छिन् Pes: چرا او می خوابد Pnb: کیوں او سوندی اے Snd: ڇو هو۶ سمهی ٿی Urd: کیوں وہ سوتی ہے

`mkQCl`	Prep -> IP -> Cl -> QCl	Eng: *with whom does she sleep* Hin: किस के साथ वह सोती है Nep: को सँग उनी सुत्छिन् Pes: با چه کسی او می خوابد Pnb: کرا دے نال او سوندی اے Snd: کھنچ سان ھوءَ سمھی ٿی Urd: کس کے ساتھ وہ سوتی ہے
`mkQCl`	IAdv -> NP -> QCl	Eng: *where is she* Hin: वह कहाँ है Nep: उनी कहाँ छिन् Pes: او کجا است Pnb: او کتھے اے Snd: ھوءَ ڪٿي اهي Urd: وہ کہاں ہے
`mkQCl`	IComp -> NP -> QCl	Eng: *who is this man* Hin: यह आदमी कौन है Nep: यो मान्छे को छ Pes: این مرد چه کسی است Pnb: اے بندہ کون اے Snd: هي ماڻهو ڪير اهي Urd: یہ آدمی کون ہے
`mkQCl`	IP -> QCl	Eng: *which city is there* Hin: वहाँ कौन सा शहर है Nep: त्यहाँ कुन शहर छ Pes: کدام شهر است Pnb: اوتھے کیڑا شہر اے Snd: Urd: وہاں کون سا شہر ہے

QS - question

Function	Type	Example
`mkQS`	Cl -> QS	Eng: *does she sleep* Hin: क्या वह सोती है Nep: के उनी सुत्छिन् Pes: آیا او می خوابد Pnb: کی او سوندی اے Snd: ڇا هوءَ سمھي ٿي Urd: کیا وہ سوتی ہے

Quant - quantifier ('nucleus' of Det)

Function	Type	Example
`mkQuant`	Pron -> Quant	Eng: *my house* Hin: मेरा घर Nep: मेरो घर Pes: خانه ی من Pnb: میرا گھر Snd: گھر Urd: میرا گھر
`no_Quant`	Quant	Eng: *no house* Hin: Nep: हैन घर Pes: Pnb: کوی نہیں گھر Snd: ڇو ن گھر Urd: کوی نہیں گھر

`that_Quant`	Quant	Eng: *that house* Hin: वह घर Nep: त्यो घर Pes: آن خانه Pnb: وہ گھر Snd: جيڪو گھر Urd: وہ گھر
`this_Quant`	Quant	Eng: *this house* Hin: यह घर Nep: यो घर Pes: این خانه Pnb: اے گھر Snd: هي گھر Urd: یہ گھر

RCl - relative clause, with all tenses

Function	Type	Example
`mkRCl`	RP -> VP -> RCl	Eng: *woman who always sleeps* Hin: औरत जो हमेशा सोती है Nep: आईमाई जो सधैँ सुत्छे Pes: زنی که همیشه می خوابد Pnb: زنانی جیڑی ہمیشہ سوندی اے Snd: استری جھڙي هميشھ سمهي ٿي Urd: عورت جو ہمیشہ سوتی ہے
`mkRCl`	RP -> V -> RCl	Eng: *woman who sleeps* Hin: औरत जो सोती है Nep: आईमाई जो सुत्छे Pes: زنی که می خوابد Pnb: زنانی جیڑی سوندی اے Snd: استری جھڙي سمهي ٿي Urd: عورت جو سوتی ہے
`mkRCl`	RP -> V2 -> NP -> RCl	Eng: *woman who loves him* Hin: औरत जो उस को प्यार करती है Nep: आईमाई जो उ लाई माया गर्छे Pes: زنی که او را دوست دارد Pnb: زنانی جیڑی اونوں پیار کردی اے Snd: استری جھڙي هو سان عشق کري ٿي Urd: عورت جو اس کو پیار کرتی ہے
`mkRCl`	RP -> VV -> VP -> RCl	Eng: *woman who wants to sleep* Hin: औरत जो सोना चाहती है Nep: आईमाई जो सुत्न चाहन्छे Pes: زنی که می خواهد بخوابد Pnb: زنانی جیڑی سونا چاندی اے Snd: استری جھڙي سمهڻ چاهي ٿي Urd: عورت جو سونا چاہتی ہے
`mkRCl`	RP -> VS -> S -> RCl	Eng: *woman who says that I sleep* Hin: औरत जो कहती है कि मैं सोता हूँ Nep: आईमाई जो भन्छेकी म सुत्छु Pes: زنی که می گوید که من می خوابم Pnb: زنانی جیڑی کہدی اے کہ میں سوندا واں Snd: استری جھڙي چوي ٿي ت مان سمهان ٿو Urd: عورت جو کہتی ہے کہ میں سوتا ہوں

mkRCl	RP -> VQ -> QS -> RCl	Eng: *woman who wonders who sleeps* Hin: औरत जो हैरान होती है कि कौन सोता है Nep: आईमाई जो अचम्म हुन्छेकी को सुत्छन् Pes: Pnb: زنانی جیڑی حیران ہوندی اے کہ کون سوندا اے Snd: استری جھڑی هيران ٿيی ٿی ت ڪير سمهی ٿو Urd: عورت جو حیران ہوتی ہے کہ کون سوتا ہے
mkRCl	RP -> VA -> A -> RCl	Eng: *woman who becomes old* Hin: औरत जो बूढ़ी बनती है Nep: आईमाई जो बुढी हुन्छे Pes: زنی که پیر می شود Pnb: زنانی جیڑی بوڈی بندی اے Snd: استری جھڑی پوڑھی ٿيی ٿی Urd: عورت جو بوڑھی بنتی ہے
mkRCl	RP -> VA -> AP -> RCl	Eng: *woman who becomes very old* Hin: औरत जो बहुत बूढ़ी बनती है Nep: आईमाई जो धेरै बुढी हुन्छे Pes: زنی که خیلی پیر می شود Pnb: زنانی جیڑی بہت بوڈی بندی اے Snd: استری جھڑی تمام پوڑھی ٿيی ٿی Urd: عورت جو بہت بوڑھی بنتی ہے
mkRCl	RP -> V2A -> NP -> A -> RCl	Eng: *woman who paints it red* Hin: औरत जो इस को लाल रंग करती है Nep: आईमाई जो यो लाई रातो रँग लागाउँछे Pes: زنی که آن را قرمز رنگ می کند Pnb: زنانی جیڑی اینوں لال رنگ کردی اے Snd: استری جھڑی هن کی ڳاڙهو رنگ کری ٿی Urd: عورت جو اس کو لال رنگ کرتی ہے
mkRCl	RP -> V2A -> NP -> AP -> RCl	Eng: *woman who paints it very red* Hin: औरत जो इस को बहुत लाल रंग करती है Nep: आईमाई जो यो लाई धेरै रातो रँग लागाउँछे Pes: زنی که آن را خیلی قرمز رنگ می کند Pnb: زنانی جیڑی اینوں بہت لال رنگ کردی اے Snd: استری جھڑی هن کی تمام ڳاڙهو رنگ کری ٿی Urd: عورت جو اس کو بہت لال رنگ کرتی ہے
mkRCl	RP -> V2S -> NP -> S -> RCl	Eng: *woman who answers to him that we sleep* Hin: औरत जो उस को जवाब देती है कि हम सोते हैं Nep: आईमाई जो उ लाई उत्तर दिन्छेकी हामीहरु सुत्छौं Pes: زنی که به او جواب می دهد که ما می خوابیم Pnb: زنانی جیڑی اونوں جواب دیندی اے کہ اسی سوندے واں Snd: استری جھڑی هو کی جواب ڏيی ٿی ت اسان سمهون ٿا Urd: عورت جو اس کو جواب دیتی ہے کہ ہم سوتے ہیں
mkRCl	RP -> V2Q -> NP -> QS -> RCl	Eng: *woman who asks him who sleeps* Hin: औरत जो उस से पूछती है कि कौन सोता है Nep: आईमाई जो उ सँग सोध्छेकी को सुत्छन् Pes: زنی که از او می پرسد چه کسی می خوابد Pnb: زنانی جیڑی اونوں پچھدی اے کہ کون سوندا اے Snd: استری جھڑی هو کان پڇی ٿی ت ڪير سمهی ٿو Urd: عورت جو اس سے پوچھتی ہے کہ کون سوتا ہے
mkRCl	RP -> V2V -> NP -> VP -> RCl	Eng: *woman who begs him to sleep* Hin: औरत जो उस से सोने की भीख मांगती है Nep: आईमाई जो उ लाई सुत्न आग्रह गर्छ Pes: زنی که از او خواهش می کند بخوابد Pnb: زنانی جیڑی اونوں سونا دی مانگدی اے Snd: Urd: عورت جو اس سے سونے کی التجا کرتی ہے

`mkRCl`	RP -> A -> RCl	Eng: *woman who is old* Hin: औरत जो बूढ़ी है Nep: आईमाई जो बुढी छे Pes: زنی که پیر است Pnb: زنانی جیڑی بوڈی اے Snd: استری جھڙی پوڙھی اھی Urd: عورت جو بوڑھی ہے
`mkRCl`	RP -> A -> NP -> RCl	Eng: *woman who is older than he* Hin: औरत जो उस से बूढ़ी है Nep: आईमाई जो उ भन्दा बुढी छे Pes: زنی که پیر تر از او است Pnb: زنانی جیڑی اورے توں بوڈی اے Snd: استری جھڙی هو کان پوڙھی اھی Urd: عورت جو اس سے بوڑھی ہے
`mkRCl`	RP -> AP -> RCl	Eng: *woman who is very old* Hin: औरत जो बहुत बूढ़ी है Nep: आईमाई जो धेरै बुढी छे Pes: زنی که خیلی پیر است Pnb: زنانی جیڑی بہت بوڈی اے Snd: استری جھڙی تمام پوڙھی اھی Urd: عورت جو بہت بوڑھی ہے
`mkRCl`	RP -> NP -> RCl	Eng: *woman who is the woman* Hin: औरत जो औरत है Nep: आईमाई जो आईमाई हुन् Pes: زنی که زن است Pnb: زنانی جیڑی زنانی اے Snd: استری جھڙی استری اھی Urd: عورت جو عورت ہے
`mkRCl`	RP -> Adv -> RCl	Eng: *woman who is here* Hin: औरत जो यहाँ है Nep: आईमाई जो यहाँ छे Pes: زنی که اینجا است Pnb: زنانی جیڑی ایتھے اے Snd: استری جھڙی هتی اھی Urd: عورت جو یہاں ہے
`mkRCl`	RP -> NP -> V2 -> RCl	Eng: *woman whom we love* Hin: औरत जिस को हम प्यार करते हैं Nep: आईमाई जो लाई हामीहरु माया गर्छौँ Pes: زنی که ما دوست داریم Pnb: زنانی جن نوں اسی پیار کردے واں Snd: استری جنهن سان اسان عشق کرون ٿا Urd: عورت جس کو ہم پیار کرتے ہیں
`mkRCl`	RP -> ClSlash -> RCl	Eng: *woman whom she loves today* Hin: औरत जिस को आज वह प्यार करती है Nep: आईमाई जो लाई आज उनी माया गर्छिन् Pes: زنی که امروز او دوست دارد Pnb: زنانی جن نوں اج او پیار کردی اے Snd: استری جنهن سان اڄ هوءَ عشق کری ٿی Urd: عورت جس کو آج وہ پیار کرتی ہے

RP - relative pronoun

Function	Type	Example
`which_RP`	RP	Eng: *which* Hin: जो Nep: जो Pes: که Pnb: جیڑا Snd: جھڑا Urd: جو

RS - relative

Function	Type	Example
`mkRS`	Conj -> RS -> RS -> RS	Eng: *woman who sleeps or whom we love* Hin: औरत जो सोती है या जिस को हम प्यार करते हैं Nep: आईमाई जो सुत्छे अथवा जो लाई हामीहरु माया गर्छौं Pes: زنی که می خوابد یا که ما دوست داریم Pnb: زنانی جیڑی سوندی اے یا جن نوں اسی پیار کردے واں Snd: استری جھڙی سمھی ٿي يا جنهن سان اسان عشق ڪرون ٿا Urd: عورت جو سوتی ہے یا جس کو ہم پیار کرتے ہیں

S - declarative sentence

Function	Type	Example
`mkS`	(Tense) -> (Ant) -> (Pol) -> Cl -> S	Eng: *she wouldn't have slept* Hin: वह न सोएगी Nep: उनी सुतेकिहुन्थि Pes: او نمی خوابید Pnb: او نا شاید سوئے Snd: هوءَ شايد ن سمهندي Urd: وہ نا شاید سوئے
`mkS`	Conj -> S -> S -> S	Eng: *she sleeps and I run* Hin: वह सोती है और मैं दौड़ता हूँ Nep: उनी सुत्छिन् र म कुद्छु Pes: او می خوابد و من می دوم Pnb: او سوندی اے تے میں دوڑدا واں Snd: هوءَ سمهي ٿي ۽ مان ڊوڙان ٿو Urd: وہ سوتی ہے اور میں دوڑتا ہوں
`mkS`	Conj -> ListS -> S	Eng: *she sleeps , I run and you walk* Hin: वह सोती है , मैं दौड़ता हूँ और तू चलता है Nep: उनी सुत्छिन् , म कुद्छु र तिमी हिड्छौ Pes: او می خوابد من می دوم و تو راه می روی Pnb: او سوندی اے میں دوڑدا واں تے توں چلدا ایں Snd: هوءَ سمهي ٿي مان ڊوڙان ٿو ۽ تون هلين ٿو Urd: وہ سوتی ہے میں دوڑتا ہوں اور تو چلتا ہے
`mkS`	Adv -> S -> S	Eng: *today she sleeps* Hin: आज वह सोती है Nep: आज उनी सुत्छिन् Pes: امروز او می خوابد Pnb: اج او سوندی اے Snd: اڄ هوءَ سمهي ٿي Urd: آج وہ سوتی ہے

SC - embedded sentence or question

Function	Type	Example
`mkSC`	S -> SC	Eng: *that she sleeps* Hin: कि वह सोती है Nep: की उनी सुत्छिन् Pes: که او می خوابد Pnb: کہ او سوندی اے Snd: ت هوء سمهی ٿی Urd: کہ وہ سوتی ہے
`mkSC`	QS -> SC	Eng: *who sleeps* Hin: कि कौन सोता है Nep: को सुत्छिन् Pes: چه کسی می خوابد Pnb: کون سوندا اے Snd: کير سمهی ٿو Urd: کہ کون سوتا ہے

SSlash

Function	Type	Example
`mkSSlash`	Temp -> Pol -> ClSlash -> SSlash	Eng: *she hadn't seen* Hin: वह नहीं देख चुकी थी Nep: उनी हेरेकिथिइनन् Pes: او ندیده بود Pnb: او نیں ویکھ گیی سی Snd: هوء ڏسيل ن هءي Urd: وہ نہیں دیکھ چکی تھی

Sub100

Function	Type	Example
`mkSub100`	Unit -> Sub100	Eng: *eight* Hin: आठ Nep: आठ Pes: هشت Pnb: اٹھ Snd: ا□ث Urd: آٹھ
`tenfoldSub100`	Unit -> Sub100	Eng: *eight* Hin: आठ Nep: आठ Pes: هشت Pnb: اٹھ Snd: ا□ث Urd: آٹھ

Sub1000

Function	Type	Example
`mkSub1000`	Unit -> Sub1000	Eng: *nine hundred* Hin: नौ सौ Nep: नौ सय Pes: نهصد Pnb: نو سو Snd: نو سو Urd: نو سو

Subj - subjunction

Function	Type	Example
`although_Subj`	Subj	Eng: *although she sleeps* Hin: अगरचि वह सोती है Nep: तैपनि उनी सुत्छिन् Pes: با وجود این که او می خوابد Pnb: پاویں او سوندی اے Snd: جيتوڻيڪ هوءَ سمهي ٿي Urd: اگرچہ وہ سوتی ہے
`because_Subj`	Subj	Eng: *because she sleeps* Hin: क्योंकि वह सोती है Nep: किनभने उनी सुत्छिन् Pes: برای این که او می خوابد Pnb: کیونکہ او سوندی اے Snd: ڇاڪاڻ تي هوءَ سمهي ٿي Urd: کیونکہ وہ سوتی ہے
`if_Subj`	Subj	Eng: *if she sleeps* Hin: अगर वह सोती है Nep: यदि उनी सुत्छिन् Pes: اگر که او می خوابد Pnb: اگر او سوندی اے Snd: جيڪڏهن هوءَ سمهي ٿي Urd: اگر وہ سوتی ہے
`that_Subj`	Subj	Eng: *that she sleeps* Hin: कि वह सोती है Nep: त्यो उनी सुत्छिन् Pes: آن که او می خوابد Pnb: کہ او سوندی اے Snd: اها هوءَ سمهي ٿي Urd: کہ وہ سوتی ہے
`when_Subj`	Subj	Eng: *when she sleeps* Hin: कब वह सोती है Nep: कहिले उनी सुत्छिन् Pes: وقتی که او می خوابد Pnb: کدوں او سوندی اے Snd: جڏهن هوءَ سمهي ٿي Urd: جب وہ سوتی ہے

Tense - tense

Function	Type	Example
`futureTense`	Tense	Eng: *she will sleep* Hin: वह सोएगी Nep: उनी सुत्लेछिन् Pes: او خواهد خوابید Pnb: او سوئے گی Snd: هوءَ سمهندی هوندی Urd: وہ سوئے گی
`pastTense`	Tense	Eng: *she slept* Hin: वह सोयी Nep: उनी सुतिन् Pes: او خوابید Pnb: او سوئی Snd: هوءَ سمهيل هئي Urd: وہ سوئی
`presentTense`	Tense	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुत्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوءَ سمهي ٿي Urd: وہ سوتی ہے

Text - text consisting of several phrases

Function	Type	Example
`mkText`	Phr -> (Punct) -> (Text) -> Text	Eng: *does she sleep ? yes .* Hin: क्या वह सोती है ? हाँ . Nep: के उनी सुत्छिन् ? हजुर . Pes: آیا او می خوابد بله □ Pnb: کی او سوندی اے ہاں . Snd: ڇا هوءَ سمهي ٿي ها . Urd: کیا وہ سوتی ہے ہاں
`mkText`	S -> Text	Eng: *she slept .* Hin: वह सोयी . Nep: उनी सुतिन् . Pes: او خوابید Pnb: او سوئی . Snd: هوءَ سمهيل هئي . Urd: وہ سوئی
`mkText`	Cl -> Text	Eng: *she sleeps .* Hin: वह सोती है . Nep: उनी सुत्छिन् . Pes: او می خوابد Pnb: او سوندی اے . Snd: هوءَ سمهي ٿي . Urd: وہ سوتی ہے
`mkText`	QS -> Text	Eng: *did she sleep ?* Hin: क्या वह सोयी ? Nep: के उनी सुतिन् ? Pes: آیا او خوابید Pnb: کی او سوئی Snd: ڇا هوءَ سمهيل هئي Urd: کیا وہ سوئی

`mkText`	Text -> Text -> Text	Eng: *where ? here . when ? now !* Hin: कहाँ ? यहाँ . कब ? अब ! Nep: कहाँ ? यहाँ . कहिले ? अहीले ! Pes: کجا اینجا کی حالا Pnb: کدوں ہون کتھے ایتھے . Snd: ڪڏهن هاڻی ڪٽی هتی . Urd: کہاں یہاں کب اب

Unit

Function	Type	Example
`n1_Unit`	Unit	Eng: *one* Hin: एक Nep: एक Pes: یک Pnb: اك Snd: هڪ Urd: ایک
`n2_Unit`	Unit	Eng: *two* Hin: दो Nep: दुई Pes: دو Pnb: دو Snd: ٻ Urd: دو
`n3_Unit`	Unit	Eng: *three* Hin: तीन Nep: तीन Pes: سه Pnb: تن Snd: ٽي Urd: تین
`n4_Unit`	Unit	Eng: *four* Hin: चार Nep: चार Pes: چهار Pnb: چار Snd: چار Urd: چار
`n5_Unit`	Unit	Eng: *five* Hin: पांच Nep: पाँच Pes: پنج Pnb: پنج Snd: پنج Urd: پانچ
`n6_Unit`	Unit	Eng: *six* Hin: छे Nep: छ Pes: شش Pnb: چھ Snd: ڇھ Urd: چھ

`n7_Unit`	Unit	Eng: *seven* Hin: सात Nep: सात Pes: هفت Pnb: ست Snd: ست Urd: سات
`n8_Unit`	Unit	Eng: *eight* Hin: आठ Nep: आठ Pes: هشت Pnb: اٹھ Snd: ا□ ٺ Urd: آٹھ
`n9_Unit`	Unit	Eng: *nine* Hin: नौ Nep: नौ Pes: نه Pnb: نو Snd: نو Urd: نو

Utt - sentence, question, word...

Function	Type	Example
`lets_Utt`	VP -> Utt	Eng: *let's sleep* Hin: आओ सोएँ Nep: सुतौं Pes: بیایید بخوابیم Pnb: آو سون Snd: اچ سمھون Urd: آو سوئیں
`mkUtt`	S -> Utt	Eng: *she slept* Hin: वह सोयी Nep: उनी सुतिन् Pes: او خوابید Pnb: او سوئی Snd: هوء سمهيل ه٤ی Urd: وہ سوئی
`mkUtt`	Cl -> Utt	Eng: *she sleeps* Hin: वह सोती है Nep: उनी सुत्छिन् Pes: او می خوابد Pnb: او سوندی اے Snd: هوء سمهی ٿی Urd: وہ سوتی ہے
`mkUtt`	QS -> Utt	Eng: *who didn't sleep* Hin: कौन नहीं सोया Nep: को सुतेनन् Pes: چه کسی نخوابید Pnb: کون نیں سویا Snd: کير سمهيل ن اهی Urd: کون نہیں سویا

`mkUtt`	QCl -> Utt	Eng: *who sleeps* Hin: कौन सोता है Nep: को सुत्छन् Pes: چه کسی می خوابد Pnb: کون سوندا اے Snd: کير سمهی ٿو Urd: کون سوتا ہے
`mkUtt`	IP -> Utt	Eng: *who* Hin: कौन Nep: को Pes: چه کسی Pnb: کون Snd: کير Urd: کون
`mkUtt`	IAdv -> Utt	Eng: *why* Hin: क्यों Nep: किन Pes: چرا Pnb: کیوں Snd: ڇو Urd: کیوں
`mkUtt`	NP -> Utt	Eng: *this man* Hin: यह आदमी Nep: यो मान्छे Pes: این مرد Pnb: اے بندہ Snd: هی ماڻهو Urd: یہ آدمی
`mkUtt`	Adv -> Utt	Eng: *here* Hin: यहाँ Nep: यहाँ Pes: اینجا Pnb: ایتھے Snd: هتی Urd: یہاں
`mkUtt`	VP -> Utt	Eng: *to sleep* Hin: सोना Nep: सुत्नु Pes: خوابیدن Pnb: سونا Snd: سمهڻ Urd: سونا
`mkUtt`	AP -> Utt	Eng: *good* Hin: अच्छा Nep: राम्रो Pes: خوب Pnb: اچھا Snd: سٺو Urd: اچھا
`mkUtt`	Card -> Utt	Eng: *five* Hin: पांच Nep: पाँच Pes: پنج Pnb: پنج Snd: پنج Urd: پانچ

`no_Utt`	Utt	Eng: *no* Hin: नहीं Nep: होईन Pes: نه Pnb: نیں Snd: نا Urd: نہیں
`yes_Utt`	Utt	Eng: *yes* Hin: हाँ Nep: हजुर Pes: بله Pnb: ہاں Snd: ها Urd: ہاں

V2 - two-place verb

Function	Type	Example
`have_V2`	V2	Eng: *to have it* Hin: इस रखना Nep: यो हुनु Pes: آن را داشتن Pnb: اینوں راکھنا Snd: هن رکڻ Urd: اس رکھنا

VP - verb phrase

Function	Type	Example
`mkVP`	V -> VP	Eng: *to sleep* Hin: सोना Nep: सुत्नु Pes: خوابیدن Pnb: سونا Snd: سمهڻ Urd: سونا
`mkVP`	V2 -> NP -> VP	Eng: *to love him* Hin: उस को प्यार करना Nep: उ लाई माया गर्नु Pes: او را دوست داشتن Pnb: اونوں پیار کرنا Snd: هو سان عشق کرڻ Urd: اس کو پیار کرنا
`mkVP`	V3 -> NP -> NP -> VP	Eng: *to send it to him* Hin: यह उस को भेजना Nep: यो उस्लाई लाई पठाउनु Pes: آن را برای او فرستادن Pnb: اینوں پیجنا اونوں Snd: Urd: یہ اس کو بھیجنا
`mkVP`	VV -> VP -> VP	Eng: *to want to sleep* Hin: सोना चाहना Nep: सुत्न चाहनु Pes: خواستن بخوابد Pnb: چانا سونا Snd: چاهڻ سمهڻ Urd: سونا چاہنا

mkVP	VS -> S -> VP	Eng: *to know that she sleeps* Hin: जानना कि वह सोती है Nep: थाहा पाउनुकी उनी सुत्छिन् Pes: دانستن که او می خوابد Pnb: جاننا Snd: ڄاڻڻ ت هوءَ سمهي ٿي Urd: جاننا کہ وہ سوتی ہے
mkVP	VQ -> QS -> VP	Eng: *to wonder who sleeps* Hin: हैरान होना कि कौन सोता है Nep: अचम्म हुनुकी को सुत्छन् Pes: Pnb: حیران ہونا Snd: هيران ٿيڻ ت ڪير سمهي ٿو Urd: حیران ہونا کہ کون سوتا ہے
mkVP	V2A -> NP -> AP -> VP	Eng: *to paint it red* Hin: इस को लाल रंग करना Nep: यो लाई रातो रँग लागाउनु Pes: آن را قرمز رنگ کردن Pnb: اینوں رنگ کرنا لال Snd: هن کي رنگ ڪرڻ ڳاڑهو Urd: اس کو لال رنگ کرنا
mkVP	V2S -> NP -> S -> VP	Eng: *to answer to him that she sleeps* Hin: उस को जवाब देना कि वह सोती है Nep: उ लाई उत्तर दिनुकी उनी सुत्छिन् Pes: به او جواب دادن که او می خوابد Pnb: اونوں جواب دینا Snd: هو کي جواب ڏيڻ ت هوءَ سمهي ٿي Urd: اس کو جواب دینا کہ وہ سوتی ہے
mkVP	V2Q -> NP -> QS -> VP	Eng: *to ask him who sleeps* Hin: उस से पूछना कि कौन सोता है Nep: उ सँग सोध्नुकी को सुत्छन् Pes: از او پرسیدن چه کسی می خوابد Pnb: اونوں پچھنا Snd: هو کان پڇڻ ت ڪير سمهي ٿو Urd: اس سے پوچھنا کہ کون سوتا ہے
mkVP	V2V -> NP -> VP -> VP	Eng: *to beg him to sleep* Hin: उस से सोने की भीख मांगना Nep: उ लाई सुत्न आग्रह गर्नु Pes: از او خواهش کردن بخوابد Pnb: اونوں مانگنا سونا دی Snd: Urd: اس سے سونے کی التجا کرنا
mkVP	Adv -> VP	Eng: *to be here* Hin: यहाँ Nep: यहाँ Pes: اینجا بودن Pnb: ایتھے Snd: هتي Urd: یہاں
mkVP	VP -> Adv -> VP	Eng: *to sleep here* Hin: यहाँ सोना Nep: यहाँ सुत्नु Pes: اینجا خوابیدن Pnb: سونا ایتھے Snd: سمهڻ هتي Urd: یہاں سونا

`mkVP`	AdV -> VP -> VP	Eng: *to always sleep* Hin: हमेशा हमेशा सोना Nep: सधैँ सुत्नु Pes: همیشه خوابیدن Pnb: سونا Snd: سمهڻ Urd: ہمیشہ ہمیشہ سونا
`mkVP`	VPSlash -> NP -> VP	Eng: *to paint it black* Hin: इस को काला रंग करना Nep: यो लाई कालो रँग लागाउनु Pes: آن را سیاه رنگ کردن Pnb: اینوں رنگ کرنا کالا Snd: هن کی رنگ ڪرڻ ڪارو Urd: اس کو کالا رنگ کرنا
`mkVP`	VPSlash -> VP	Eng: *to paint itself black* Hin: ख़ुद को काला रंग करना Nep: आफैं लाई कालो रँग लागाउनु Pes: خودش را سیاه رنگ کردن Pnb: رنگ کرنا خود نوں کالا Snd: رنگ ڪرڻ پاڻ کی ڪارو Urd: خود کو کالا رنگ کرنا
`passiveVP`	V2 -> VP	Eng: *to be loved* Hin: प्यार करना Nep: माया गर्नु Pes: دوست داشتن Pnb: پیار کرنا Snd: عشق ڪرڻ Urd: پیار کرنا
`passiveVP`	V2 -> NP -> VP	Eng: *to be loved by her* Hin: उस से प्यार करना Nep: उनी लाइ माया गर्नु Pes: توسط او دوست داشتن Pnb: پیار کرنا اونوں Snd: عشق ڪرڻ هو هٿان Urd: اس سے پیار کرنا
`progressiveVP`	VP -> VP	Eng: *to be sleeping* Hin: सोना Nep: सुतदै Pes: خوابیدن Pnb: سونا Snd: سمهڻ Urd: سونا
`reflexiveVP`	V2 -> VP	Eng: *to love itself* Hin: ख़ुद को प्यार करना Nep: आफैं लाई माया गर्नु Pes: خودش را دوست داشتن Pnb: پیار کرنا خود نوں Snd: عشق ڪرڻ پاڻ سان Urd: خود کو پیار کرنا

VPSlash - verb phrase missing complement

Function	Type	Example
`mkVPSlash`	V2 -> VPSlash	Eng: *whom does she see* Hin: किस को वह देखती है Nep: कासलाई उनी हेर्छिन् Pes: چه کسی را او می بیند Pnb: کون او ویکھدی اے Snd: کير هوء ڏسي ٿي Urd: کس کو وہ دیکھتی ہے
`mkVPSlash`	V2A -> AP -> VPSlash	Eng: *whom does she paint red* Hin: किस को वह लाल रंग करती है Nep: कासलाई उनी रातो रँग लागाउँछिन् Pes: چه کسی را او قرمز رنگ می کند Pnb: کون او لال رنگ کردی اے Snd: کير هوء ڳاڙهو رنگ ڪري ٿي Urd: کس کو وہ لال رنگ کرتی ہے
`mkVPSlash`	V2Q -> QS -> VPSlash	Eng: *whom does she ask where I sleep* Hin: किस को वह पूछती है कि कहाँ मैं सोता हूँ Nep: कासलाई उनी सोध्छिन्की कहाँ म सुत्छु Pes: از چه کسی او می پرسد کجا من می خوابم Pnb: کون او پچھدی اے کہ کتھے میں سوندا واں Snd: کير هوء پڇي ٿي ت ڪٿي مان سمهان ٿو Urd: کس کو وہ پوچھتی ہے کہ کہاں میں سوتا ہوں
`mkVPSlash`	V2S -> S -> VPSlash	Eng: *to whom does she answer that I sleep* Hin: किस को वह जवाब देती है कि मैं सोता हूँ Nep: कासलाई उनी उत्तर दिन्छिन्की म सुत्छु Pes: به چه کسی او جواب می دهد که من می خوابم Pnb: کون او جواب دیندی اے کہ میں سوندا واں Snd: کير هوء جواب ڏيي ٿي ت مان سمهان ٿو Urd: کس کو وہ جواب دیتی ہے کہ میں سوتا ہوں
`mkVPSlash`	V2V -> VP -> VPSlash	Eng: *whom does she beg to sleep* Hin: किस को वह सोने की भीख मांगती है Nep: कासलाई उनी सुत्न आग्रह गर्छिन् Pes: از چه کسی او خواهش می کند بخوابد Pnb: کون او سونا دی مانگدی اے Snd: Urd: کس کو وہ سونے کی التجا کرتی ہے
`mkVPSlash`	VV -> VPSlash -> VPSlash	Eng: *whom does she want to see* Hin: किस को वह देखना चाहती है Nep: कासलाई उनी हेर्न चाहन्छिन् Pes: چه کسی را او می خواهد ببیند Pnb: کون او ویکھنا چاندی اے Snd: کير هوء ڏسڻ چاهي ٿي Urd: کس کو وہ دیکھنا چاہتی ہے

VV - verb-phrase-complement verb

Function	Type	Example
`can_VV`	VV	Eng: *to be able to sleep* Hin: सो सकना Nep: सुत्न सक्नु Pes: Pnb: سکنا سون Snd: سگهڻ سمهن Urd: سو سکنا

want_VV	VV	Eng: *to want to sleep* Hin: सोना चाहना Nep: सुत्न चाहनु Pes: خواستن بخوابد Pnb: چانا سونا Snd: چاهڻ سمهڻ Urd: سونا چاہنا

Lexical Paradigms

Paradigms for English

source `../src/english/ParadigmsEng.gf`

Function	Type	Explanation
`Gender`	Type	-
`human`	Gender	-
`nonhuman`	Gender	-
`Number`	Type	-
`singular`	Number	-
`plural`	Number	-
`npNumber`	NP -> Number	*exctract the number of a noun phrase*
`mkN`	`(flash : Str)` -> N	*plural s, incl. flash-flashes, fly-flies*
`mkN`	`(man,men : Str)` -> N	*irregular plural*
`mkN`	`(man,men,man's,men's : Str)` -> N	*irregular genitives*
`mkN`	Gender -> N -> N	*default nonhuman*
`mkN`	`Str` -> N -> N	*e.g. baby + boom*
`mkN2`	N -> N2	*e.g. wife of (default prep. to)*
`mkN2`	N -> Prep -> N2	*e.g. access to*
`mkN3`	N -> Prep -> Prep -> N3	*e.g. connection from x to y*
`mkPN`	`Str` -> PN	-
`mkA`	`(happy : Str)` -> A	*regular adj, incl. happy-happier, rude-ruder*
`mkA`	`(fat,fatter : Str)` -> A	*irreg. comparative*
`mkA`	`(good,better,best,well : Str)` -> A	*completely irreg.*
`compoundA`	A -> A	*force comparison with more/most*
`simpleA`	A -> A	*force comparison with -er,-est*
`irregAdv`	A -> `Str` -> A	*adverb irreg, e.g. "fast"*
`mkA2`	A -> Prep -> A2	*absent from*
`mkAdv`	`Str` -> Adv	*e.g. today*
`mkAdV`	`Str` -> AdV	*e.g. always*
`mkAdA`	`Str` -> AdA	*e.g. quite*
`mkAdN`	`Str` -> AdN	*e.g. approximately*
`mkPrep`	`Str` -> Prep	*e.g. "in front of"*
`noPrep`	Prep	*no preposition*
`mkV`	`(cry : Str)` -> V	*regular, incl. cry-cries, kiss-kisses etc*
`mkV`	`(stop, stopped : Str)` -> V	*reg. with consonant duplication*
`mkV`	`(drink, drank, drunk : Str)` -> V	*ordinary irregular*
`mkV`	`(go, goes, went, gone, going : Str)` -> V	*totally irregular*
`mkV`	`Str` -> V -> V	*fix compound, e.g. under+take*
`partV`	V -> `Str` -> V	*with particle, e.g. switch + on*
`reflV`	V -> V	*reflexive e.g. behave oneself*
`mkV2`	V -> V2	*transitive, e.g. hit*
`mkV2`	V -> Prep -> V2	*with preposiiton, e.g. believe in*
`mkV3`	V -> V3	*ditransitive, e.g. give,_,_*
`mkV3`	V -> Prep -> Prep -> V3	*two prepositions, e.g. speak, with, about*
`mkVS`	V -> VS	*sentence-compl e.g. say (that S)*
`mkV2S`	V -> Prep -> V2S	*e.g. tell (NP) (that S)*

mkVV	V -> VV	*e.g. want (to VP)*
ingVV	V -> VV	*e.g. start (VPing)*
mkV2V	V -> Prep -> Prep -> V2V	*e.g. want (noPrep NP) (to VP)*
ingV2V	V -> Prep -> Prep -> V2V	*e.g. prevent (noPrep NP) (from VP-ing)*
mkVA	V -> VA	*e.g. become (AP)*
mkV2A	V -> Prep -> V2A	*e.g. paint (NP) (AP)*
mkVQ	V -> VQ	*e.g. wonder (QS)*
mkV2Q	V -> Prep -> V2Q	*e.g. ask (NP) (QS)*

Paradigms for Hindi/Urdu

source `../src/urdu/ParadigmsUrd.gf`

Function	Type	Explanation
masculine	Gender	-
feminine	Gender	-
singular	Number;	-
plural	Number;	-
mkN	Str -> N	*Regular nouns e.g. laRka, gender is judged from noun ending*
mkN	Str -> Gender -> N	*nouns whose gender is irregular e.g. aadmy*
mkN	(x1,_,_,_,_,x6 : Str) -> Gender -> N	*worst case constructor*
mkN2	N -> Prep -> Str -> N2;	*e.g. maN ky*
mkN3	N -> Prep -> Str -> Str-> N3	*e.g. faSlh - sE - ka*
mkCmpdNoun	Str -> N -> N	*e.g. talab elm*
mkPN	Str -> PN	*e.g. John*
demoPN	Str -> Str -> Str -> Quant	-
mkDet	Str -> Str -> Str -> Str -> Number -> Det	-
mkIP	(x1,x2,x3:Str) -> Number -> Gender -> IP	-
mkAdN	Str -> AdN	-
mkA	Str-> A	*e.g accha*
mkA	Str -> Str -> A2	*e.g sE Xady krna*
mkA2	A -> Str -> A2	-
mkCompoundA	Str -> Str -> A	*e.g dra hwa*
mkV	Str -> V	*regular verbs e.g. sona*
mkV2	Str -> V2	*e.g. pyna*
mkV2	V -> V2	*e.g. pyna*
mkV2	V -> Str -> V2	*e.g. bnd krna*
dirV2	V -> V2	-
mkV3	V -> Str -> Str -> V3;	*e.g. bycna*
mkV2V	V -> Str -> Str -> Bool -> V2V	*e.g. eltja krna - sE - kw*
dirdirV3	V -> V3	-
compoundV	Str -> V -> V	*e.g. barX hwna*
compoundV	Str -> V2 -> V	*e.g. bnd krna*
mkAdv	Str -> Adv	*e.g yhaN*
mkAdv	Str -> Str -> Adv	-
mkPrep	Str -> Str -> Prep	*e.g. ka - ky*
mkIQuant	Str -> IQuant	-
mkConj	Str -> Conj	*and (plural agreement)*
mkConj	Str -> Number -> Conj	*or (agrement number given as argument)*
mkConj	Str -> Str -> Conj	*both ... and (plural)*
mkConj	Str -> Str -> Number -> Conj	*either ... or (agrement number given as argument)*
mkConj	Str -> Conj	-
mkConj	Str -> Number -> Conj	-
mkConj	Str -> Str -> Conj	-
mkConj	Str -> Str -> Number -> Conj	-
mk2Conj	Str -> Str -> Number -> Conj	-

mkVS	V -> VS;	*e.g drna*
mkVV	V -> VV	-
mkAdA	Str -> AdA	-
mkVQ	V -> VQ	*e.g. janna*

Paradigms for Punjabi

source ../src/punjabi/ParadigmsPnb.gf

Function	Type	Explanation
masculine	Gender	-
feminine	Gender	-
singular	Number;	-
plural	Number;	-
mkN	Str -> N	*Regular nouns e.g. 'munda', gender is judged from noun ending*
mkN2	N -> Prep -> Str -> N2;	*e.g. maN da*
mkCmpdNoun	Str -> N -> N	-
mkPN	Str -> PN	*e.g. John*
mkPN	Str -> Gender -> PN	-
demoPN	Str -> Str -> Str -> Quant	-
mkDet	Str -> Str -> Str -> Str -> Number -> Det	-
mkIP	(x1,x2,x3,x4:Str) -> Number -> Gender -> IP	-
mkAdN	Str -> AdN	-
mkA	Str-> A	*e.g. changa*
mkA	Str -> Str -> A2	-
mkV	Str -> V	*e.g. saona*
mkV2	Str -> V2	*e.g. khana*
mkV2	V -> V2	-
mkV2	V -> Str -> V2	-
mkV3	V -> Str -> Str -> V3;	*e.g. vechna*
mkV2V	V -> Str -> Str -> Bool -> V2V	*e.g. mangna*
compoundV	Str -> V -> V	*e.g. bnd krna*
compoundV	Str -> V2 -> V	-
mkAdv	Str -> Adv	*e.g. aj*
mkPrep	Str -> Prep	*e.g da*
mkQuant1	Pron -> Quant	-
mkIQuant	Str -> Str -> Str -> Str -> IQuant	-
mkQuant1	Pron -> Quant	-
mkConj	Str -> Conj	*and (plural agreement)*
mkConj	Str -> Number -> Conj	*or (agrement number given as argument)*
mkConj	Str -> Str -> Conj	*both ... and (plural)*
mkConj	Str -> Str -> Number -> Conj	*either ... or (agrement number given as argument)*
mkConj	Str -> Conj	-
mkConj	Str -> Number -> Conj	-
mkConj	Str -> Str -> Conj	-
mkConj	Str -> Str -> Number -> Conj	-
mk2Conj	Str -> Str -> Number -> Conj	-

Paradigms for Persian

source ../src/persian/ParadigmsPes.gf

Function	Type	Explanation
animate	Animacy	-
inanimate	Animacy	-

singular	Number;	-
plural	Number;	-
mkN01	Str -> Animacy -> Noun	-
mkN02	Str -> Animacy -> Noun	-
mkN2	N -> Prep -> Str -> N2;	-
mkN3	N -> Prep -> Str -> Str-> N3	-
mkCmpdNoun1	Str -> N -> N	-
mkCmpdNoun2	N -> Str -> N	-
mkPN	Str -> Animacy -> PN	-
personalPN	Str -> Number -> PPerson -> Pron	-
demoPN	Str -> Str -> Str -> Quant	-
mkDet	Str -> Number -> Det	-
mkDet	Str -> Number -> Bool -> Det	-
mkIP	(x1,x2,x3,x4:Str) -> Number -> Gender -> IP	-
mkAdN	Str -> AdN	-
mkA	Str-> A	-
mkA	Str-> Str -> A	-
mkA	Str -> Str -> A2	-
mkV	Str -> Str -> V	-
haveVerb	V	-
mkV_1	Str -> V	-
mkV_2	Str -> V	-
mkV2	V -> V2	-
mkV2	V -> Str -> V2	-
mkV2	V -> Str -> Bool -> V2	-
mkV3	V -> Str -> Str -> V3;	-
mkV2V	V -> Str -> Str -> Bool -> V2V	-
compoundV	Str -> V -> V	-
compoundV	Str -> V2 -> V	-
mkAdv	Str -> Adv	-
mkPrep	Str -> Prep	-
mkQuant	Str -> Str -> Quant	-
mkConj	Str -> Conj	*and (plural agreement)*
mkConj	Str -> Number -> Conj	*or (agrement number given as argument)*
mkConj	Str -> Str -> Conj	*both ... and (plural)*
mkConj	Str -> Str -> Number -> Conj	*either ... or (agrement number given as argument)*
mkConj	Str -> Conj	-
mkConj	Str -> Number -> Conj	-
mkConj	Str -> Str -> Conj	-
mkConj	Str -> Str -> Number -> Conj	-
mk2Conj	Str -> Str -> Number -> Conj	-
mkVV	V -> VV	-

Paradigms for Sindhi

source `../src/sindhi/ParadigmsSnd.gf`

Function	Type	Explanation
masculine	Gender	-
feminine	Gender	-
singular	Number;	-
plural	Number;	-
mkN2	N -> Prep -> Str -> N2;	-
mkN3	N -> Prep -> Str -> Str-> N3	-
mkCmpdNoun	Str -> N -> N	-
mkPN	Str -> PN	-
mkPN	Str -> Gender -> PN	-

demoPN	Str -> Str -> Str -> Quant	-
mkDet	Str -> Str -> Str -> Str -> Number -> Det	-
mkIP	(x1,x2,x3,x4:Str) -> Number -> Gender -> IP	-
mkAdN	Str -> AdN	-
mkA	Str-> A	-
mkA	Str -> Str -> A2	-
mkV	Str -> V	-
mkV2	Str -> V2	-
mkV2	V -> V2	-
mkV2	V -> Str -> V2	-
mkV3	V -> Str -> Str -> V3;	-
mkV2V	V -> Str -> Str -> Bool -> V2V	-
compoundV	Str -> V -> V	-
compoundV	Str -> V2 -> V	-
mkAdv	Str -> Adv	-
mkPrep	Str -> Prep	-
mkQuant1	Pron -> Quant	-
mkIQuant	Str -> Str -> Str -> Str -> IQuant	-
mkQuant1	Pron -> Quant	-
mkConj	Str -> Conj	*and (plural agreement)*
mkConj	Str -> Number -> Conj	*or (agrement number given as argument)*
mkConj	Str -> Str -> Conj	*both ... and (plural)*
mkConj	Str -> Str -> Number -> Conj	*either ... or (agrement number given as argument)*
mkConj	Str -> Conj	-
mkConj	Str -> Number -> Conj	-
mkConj	Str -> Str -> Conj	-
mkConj	Str -> Str -> Number -> Conj	-
mk2Conj	Str -> Str -> Number -> Conj	-
mkVV	V -> VV	-

Paradigms for Nepali

source `../src/nepali/ParadigmsNep.gf`

Function	**Type**	**Explanation**
masculine	Gender	-
feminine	Gender	-
singular	Number	-
plural	Number	-
human	NType	-
profession	NType	-
living	NType	-
regN	Str -> N	-
regN	Str -> NPerson -> N	-
regN	Str -> NType -> N	-
regN	Str -> NType -> NPerson -> N	-
mkNF	Str -> N	-
mkNF	Str -> NPerson -> N	-
mkNF	Str -> NType -> N	-
mkNF	Str -> NType -> NPerson -> N	-
mkNUC	Str -> N	-
mkNUC	Str -> Gender -> N	-
mkNUC	Str -> Gender -> NType -> N	-
mkNUC	Str -> Gender -> NType -> NPerson -> N	-
mkNUC	Str -> NType -> Gender -> N	-
mkN2	N -> Prep -> Str -> N2	-
mkN2	N -> Prep -> Str -> NType -> N2	-

mkN2	N -> Prep -> Str -> NType -> NPerson -> N2	-
mkN2	N -> Prep -> Str -> N2;	-
mkN3	N -> Prep -> Prep -> Str-> N3	-
mkN3	N -> Prep -> Prep -> Str-> NType -> N3	-
mkN3	N -> Prep -> Prep -> Str-> N3	-
mkCmpdNoun	Str -> N -> N	-
mkPN	Str -> PN	-
mkPN	Str -> Gender -> NPerson -> PN	-
mkPN	Str -> Gender -> NType -> NPerson -> PN	-
demoPN	Str -> Str -> Str -> Quant	-
mkDet	(s1,s2:Str) -> Number -> Det	-
mkDet	(s1,s2,s3,s4:Str) -> Number -> Det	-
mkIDetn	(s1,s2:Str) -> Number -> IDet	-
mkIP	(x1,x2,x3,x4:Str) -> Number -> IP	-
mkA	Str-> A	-
mkA	Str -> Str -> A2	-
mkV	Str -> V	-
mkV2	Str -> V2	-
mkV2	V -> V2	-
mkV2	V -> Str -> V2	-
mkV3	V -> Str -> Str -> V3	-
mkV2V	V -> Str -> Str -> Bool -> V2V	-
compoundV	Str -> V -> V	-
compoundV	Str -> V2 -> V	-
mkAdv	Str -> Adv	*e.g. today*
mkAdV	Str -> AdV	*e.g. always*
mkAdA	Str -> AdA	*e.g. quite*
mkAdN	Str -> AdN	*e.g. approximately*
mkPrep	Str -> Prep	-
noPrep	Prep	-
--mkQuant	Pron -> Quant	-
mkQuant	(s1,s2,s3,s4:Str) -> Quant	-
mkQuant	(s1,s2:Str) -> Quant	-
mkConj	Str -> Conj	*and (plural agreement)*
mkConj	Str -> Number -> Conj	*or (agrement number given as argument)*
mkConj	Str -> Str -> Conj	*both ... and (plural)*
mkConj	Str -> Str -> Number -> Conj	*either ... or (agrement number given as argument)*
mkConj	Str -> Conj	-
mkConj	Str -> Number -> Conj	-
mkConj	Str -> Str -> Conj	-
mkConj	Str -> Str -> Number -> Conj	-
mk2Conj	Str -> Str -> Number -> Conj	-

Bibliography

[Agnihotri, 2007] Agnihotri, R. K. (2007). *Hindi: An Essential Grammar*. London/New York: Routledge.

[Ahsan et al., 2010] Ahsan, A., Kolachina, P., Kolachina, S., Sharma, D. M., and Sangal, R. (2010). Coupling statistical machine translation with rule-based transfer and generation. In *The Ninth Conference of the Association for Machine Translation in the Americas*.

[Angelov, 2011] Angelov, K. (2011). *The Mechanics of the Grammatical Framework*. PhD thesis, Chalmers University Of Technology. ISBN 978-91-7385-605-8.

[Angelov and Ranta, 2009] Angelov, K. and Ranta, A. (2009). Implementing Controlled Languages in GF. In *CNL-2009, Controlled Natural Language Workshop, Marettimo, Sicily, 2009*.

[Attempto, 2008] Attempto (2008). Attempto Project Homepage. http://attempto.ifi.uzh.ch/site/.

[Bahrani et al., 2011] Bahrani, M., Sameti, H., and Manshadi, M. H. (December 2011). A computational grammar for persian based on gpsg. *Lang Resources & Evaluatio*, Volume 45(Issue 4):387–408.

[Bender and Flickinger, 2005] Bender, E. and Flickinger, K. (2005). Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos)*, Jeju Islan, Korea.

[Bhattacharyya, 2010] Bhattacharyya, P. (2010.). Indowordnet. In *Lexical Resources Engineering Conference 2010 (LREC 2010)*, Malta.

[Blake, 1990] Blake, B. (1990). *Relational Grammar*. Routledge, London.

[Bringert et al., 2011] Bringert, B., Hallgren, T., and Ranta., A. (2011). Gf resource grammar library synopsis. www.grammaticalframework.org/lib/doc/synopsis.html.

[Butt, 1993] Butt, M. (1993). *The Structures of Complex Predicate in Hindi Stanford*. PhD thesis, stanford university.

[Butt et al., 2002] Butt, M., Dyvik, H., King, T., Masuichi, H., and Rohrer, C. (2002). The parallel gramar project. In *COLING 2002, Workshop on Grammar Engineering and Evaluation*, pages pp. 1–7.

[Butt and King, 2007] Butt, M. and King, T. (2007). Urdu in a parallel grammar development environment. *In T. Takenobu and C.-R. Huang (eds.) Language Resources and Evaluation: Special Issue on Asian Language Processing: State of the Art Resources and Processing*, 41:191–207.

[Caprotti, 2006] Caprotti, O. (2006). WebALT! Deliver Mathematics Everywhere. In *Proceedings of SITE 2006. Orlando March 20-24*. `http://webalt.math.helsinki.fi/content/e16/e301/e512/PosterDemoWebALT_eng.pdf`.

[Caprotti and Saludes, 2012] Caprotti, O. and Saludes, J. (2012). The gf mathematical grammar library. In *Conference on Intelligent Computer Mathematics /OpenMath Workshop*.

[Carbonell et al., 1992] Carbonell, J., Mitamura, T., and Nuberg, E. (1992). The kant perspective: A critique of pure transfer (and pure interlingua, pure statistics, ...). In *In the Proceedings of the Forth International Conference on Theoretical and Methodological Issues in Machine Translatio of Natural Languages*, pages 225–235.

[Carl and Ivan, 1994] Carl, P. and Ivan, A. (1994). Head-driven phrase structure grammar. *Chicago: University of Chicago Press.*

[Census-India, 2001] Census-India (2001). *Abstract of Speakers' Strength of Languages and Mother Tounges.* Government of India. http://www.censusindia.gov.in.

[Census-Pakistan, 1998] Census-Pakistan (1998). *Population by Mother Toung.* http://www.census.gov.pk/MotherTongue.htm.

[Chand, 1944] Chand, T. (1944). *The problem of Hindustani. Allahabad: Indian Periodicals.* www.columbia.edu/itc/mealac/pritchett/00fwp/sitemap.html.

[Chomsky, 1957] Chomsky, N. (1957). *Syntactic Structures.* The Hague:Mounton.

[Collins, 1997] Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 16–23, Stroudsburg, PA, USA. Association for Computational Linguistics. http://dx.doi.org/10.3115/979617.979620.

[Copestake, 2002] Copestake, A. (2002). Implementing typed feature structure grammars. *CSLI Publications.* The LKB grammars development system widely used for HPSG.

[Curry, 1961] Curry, H. B. (1961). Some logical aspects of grammatical structure. In Jakobson, R., editor, *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, pages 56–68. American Mathematical Society.

[Dalrymple, 2001] Dalrymple, M. (2001). Lexical functional grammar. *Syntax and Semantics Series. New York: Academic Press.*, (42). ISBN 0126135347.

[de Melo and Weikum, 2009] de Melo, G. and Weikum, G. (2009). Towards a universal wordnet by learning from combined evidence. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009)*, pages 513–522, New York, NY, USA. ACM.

[Détrez and Ranta, 2012] Détrez, G. and Ranta, A. (2012). Smart paradigms and the predictability and complexity of inflectional morphology. In *EACL*, pages 645–653.

[Devi, 2012] Devi, J. (2012). Implementing gf resource grammar for sindhi - a subtitle that can be quite long if necessary.

[Dorr et al., 1998] Dorr, B. J., Jordan, P. W., and Benoit, J. W. (1998). A survey of current paradigms in machine translation.

[EuroWordNet, 1996] EuroWordNet (1996). Eurowordnet. http://www.illc.uva.nl/EuroWordNet/.

[Flagship, 2012] Flagship (2012). *Undergraduate program and resource center for Hindi-Urdu at the University of Texas at Austin.* http://hindiurduflagship.org/about/two-languages-or-one/.

[Forsberg and Ranta, 2004] Forsberg, M. and Ranta, A. (2004). Functional Morphology. In *ICFP 2004, Showbird, Utah*, pages 213–223.

[Gazdar et al., 1985] Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

[Giménez and Màrquez, 2010] Giménez, J. and Màrquez, L. (2010). Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, I(94):77–86.

[Grishman and Kosaka, 1992] Grishman, R. and Kosaka, M. (1992). Combining rationalist and empiricist approaches to machine translation. In *In the Proceedings of the Forth International Conference on Theoretical and Methodological Issues in Machine Translatio of Natural Languages*, pages 263–274.

[Huet, 2005] Huet, G. (2005). A functional toolkit for morphological and phonological processing, application to a sanskrit tagger. *Journal of Functional Programming*, 15:573–614. http://journals.cambridge.org/article_S0956796804005416.

[Humayoun, 2006] Humayoun, M. (2006). Urdu Morphology, Orthography and Lexicon Extraction. MSc Thesis, Department of Computing Science, Chalmers University of Technology.

[Humayoun and Ranta, 2010] Humayoun, M. and Ranta, A. (2010.). Developing punjabi morphology, corpus and lexicon. In *The 24th Pacific Asia conference on Language, Information and Computation.*, pages pp.163–172.

[Hussain, 2004] Hussain, S. (2004). Urdu localization project. In *COLING:WORKSHOP ON Computational Approaches to Arabic Script-based Languages*, pages pp. 80–81.

[Jha et al., 2001] Jha, S., Narayan, D., Pande, P., and Bhattacharyya, P. (2001). A wordnet for hindi. In *International Workshop on Lexical Resources in Natural Language Processing*, Hyderabad, India.

[Johannisson, 2005] Johannisson, K. (2005). *Formal and Informal Software Specifications*. PhD thesis, Computer Science, Göteborg University.

[Joshi, 2012] Joshi, M. M. (2012). Save urdu from narrow minded politics. *Bombay: The Times of India*.

[Kachru, 2006] Kachru, Y. (2006). *Hindi (London Oriental and African Language Library).* Philadelphia: John Benjamins Publ. Co.

[Kale, 1894] Kale, M. R. (1894). *A Higher Sanskrit Grammar, Delhi: Motilal Banarsidass.* W. H. Allen & co.

[Kaljurand and Kuhn, 2013] Kaljurand, K. and Kuhn, T. (2013). A multilingual semantic wiki based on Attempto Controlled English and Grammatical Framework. In *Proceedings of the 10th Extended Semantic Web Conference (ESWC 2013).* Springer.

[Khalid et al., 2009] Khalid, U., Karamat, N., Iqbal, S., and Hussain, S. (2009). Semi-automatic lexical functional grammar development. In *Proceedings of the Conference on Language & Technology.*

[Lehal., 2009] Lehal., G. S. (2009.). A survey of the state of the art in punjabi language processing. *Language In India*, Volume 9(No. 10):pp. 9–23.

[Lindén and Carlson, 2010] Lindén, K. and Carlson, L. (2010). Finnwordnet – wordnet pá finska via översättning. *LexicoNordica – Nordic Journal of Lexicography*, 17:119–140.

[Ljunglöf, 2004] Ljunglöf, P. (2004). *The Expressivity and Complexity of Grammatical Framework.* PhD thesis, Dept. of Computing Science, Chalmers University of Technology and Gothenburg University. http://www.cs.chalmers.se/ peb/pubs/p04-PhD-thesis.pdf.

[Mahootiyan, 1997] Mahootiyan, S. (1997). *Persian.* Routledge.

[Martin-Löf, 1982] Martin-Löf, P. (1982). Constructive mathematics and computer programming. In Cohen, Los, Pfeiffer, and Podewski, editors, *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland, Amsterdam.

[Masica, 1991] Masica, C. (1991). *The Indo-Aryan Languages.* Cambridge, Cambridge University Press.

[Megerdoomian, 2000] Megerdoomian, K. (2000). Persian computational morphology: A unification-based approach memoranda in computer and cognitive science. Computing Research Laboratory New Mexico State University Las Cruces, New Mexico. (http://www.zoorna.org/papers/MCCS320.pdf) (Last accessed June 2011).

[Meurers et al., 2002] Meurers, W. D., Penn, G., and Richterin, F. (2002). A web-based instructional platform for constraint-based grammar formalisms and parsing,. In *Proceedings of the Effective Tools and Methodologies for Teaching NLP and CL*, page pp. 18–25.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

[Monier-Williams, 1846] Monier-Williams, M. (1846). *A Practical Grammar Of The Sanskrit Language Arranged With Reference To The Classical Languages Of Europe For The Use Of English Students*. W. H. Allen & co.

[Montague, 1974] Montague, R. (1974). *Formal Philosophy*. Yale University Press, New Haven. Collected papers edited by Richmond Thomason.

[Müller and Ghayoomi, 2010] Müller, S. and Ghayoomi, M. (2010). Pergram: A trale implementation of an hpsg fragment of persian. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, page pp. 461–467. ISBN 978-83-60810-22-4,ISSN 1896-7094.

[Naim, 1999] Naim, C. (1999). *Introductory Urdu, 2 volumes. Revised 3rd edition*. Chicago: University of Chicago.

[NCERT, 2012] NCERT (2012). *Mathematics textbooks (English and Hindi)*. New Delhi: National Council for Educational Research and Training.

[Paauw, 2009] Paauw, S. (2009). One land, one nation, one language: An analysis of indonesia's national language policy. *University of Rochester Working Papers in the Language Sciences*, 5(1):2– 16.

[Ranta, 2004] Ranta, A. (2004). Grammatical Framework: A Type-Theoretical Grammar Formalism. *The Journal of Functional Programming*, 14(2):145–189. http://www.cse.chalmers.se/ aarne/articles/gf-jfp.pdf.

[Ranta, 2009a] Ranta, A. (2009a). Grammars as Software Libraries. In Bertot, Y., Huet, G., Lévy, J.-J., and Plotkin, G., editors, *From Semantics to Computer Science. Essays in Honour of Gilles Kahn*, pages 281–308. Cambridge University Press. http://www.cse.chalmers.se/ aarne/articles/libraries-kahn.pdf.

[Ranta, 2009b] Ranta, A. (2009b). The GF Resource Grammar Library. *Linguistics in Language Technology*, 2. http://elanguage.net/journals/index.php/lilt/article/viewFile/214/158.

[Ranta, 2011] Ranta, A. (2011). *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[Ranta and Angelov, 2010] Ranta, A. and Angelov, K. (2010). Implementing Controlled Languages in GF. In *Proceedings of CNL-2009, Athens*, volume 5972 of *LNCS*, pages 82–101.

[Ranta et al., 2012] Ranta, A., Détrez, G., and Enache, R. (2012). Controlled language for everyday use: the molto phrasebook. In *CNL 2012: Controlled Natural Language*, volume 7175 of *LNCS/LNAI*.

[Rayner et al., 2000] Rayner, M., Carter, D., Bouillon, P., Digalakis, V., and Wiren., M. (2000). *The spoken Language Translator*. Cambridge University Press.

[Rizvi, 2007] Rizvi, M. (2007). *Development of Algorithms and Computational Grammar for Urdu*. PhD thesis, Pakistan Institute of Engineering and Applied Sciences, Nilore, Pakistan.

[Samvelian, 2007] Samvelian, P. (2007). A (phrasal) affix analysis of the persian ezafe,. *Journal of Linguistics*, vol. 43:pp. 605–645.

[Sarfraz and Naseem, 2007] Sarfraz, H. and Naseem, T. (2007). Sentence segmentation and segment re-ordering for english to urdu machine translation. In *In Proceedings of the Conference on Language and Technology, University of Peshawar, Pakistan.*

[Sarwat, 2006] Sarwat, R. (2006). *Language Hybridization in Pakistan (PhD thesis).* Islamabad: National University of Modern Languages.

[Schmidt, 1999] Schmidt, R. L. (1999). *Urdu an Essential Grammar*. Routledge Grammars.

[Schmidt, 2004] Schmidt, R. L. (2004). *Urdu: An Essential Grammar*. London/ New York: Routledge.

[Schuler, 2005] Schuler, K. K. (January, 2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. PhD thesis, Department of Computer and Information Science, University of Colorado.

[Seki et al., 1991] Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

[Shafqat et al., 2011] Shafqat, M., Humayoun, M., and Aarne, R. (2011). An open source punjabi resource grammar. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 70–76, Hissar, Bulgaria. RANLP 2011 Organising Committee. http://aclweb.org/anthology/R11-1010.

[Shafqat et al., 2010] Shafqat, M., Humayoun, M., and Ranta, A. (2010). An open source urdu resource grammar. In *Proceedings of the Eighth Workshop on Asian Language Resouces*, pages 153–160, Beijing, China. Coling 2010 Organizing Committee. http://www.aclweb.org/anthology/W10-3220.

[Simkhada, 2012] Simkhada, D. (2012). Implementing the gf resource grammar for nepali language. 66.

[Sinha and Mahesh, 2009] Sinha, R. and Mahesh, K. (2009). Developing english-urdu machine translation via hind. In *Third Workshop on Computational Approaches to Arabic Script-based Languages (CAASL3) in conjunction with The twelfth Machine Translation Summit.*

[Snell and Weightman, 2003] Snell, R. and Weightman, S. (2003). *Teach Yourself Hindi.* London: Hodder Education Group.

[Thurmair, 2004] Thurmair, G. (2004). Comparing rule-based and statistical mt output. In *Workshop on the amazing utility of parallel and comparable corpora.*

[Verma, 1974] Verma, M. K. (1974). The structure of the noun phrase in english and hindi by review author(s): R. k. barz, l. a. *Schwarzschild Journal of the American Oriental Society*, Vol. 94(No. 4):pp. 492–494.

[Zafar and Masood, 2009] Zafar, M. and Masood, A. (2009). Interactive english to urdu machine translation using example-based approach. *International Journal on Computer Science and Engineering*, Vol.1(3):pp 275–282.

[Zhong and Ng, 2010] Zhong, Z. and Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of the ACL 2010 System Demonstrations*, pages 78–83, Uppsala, Sweden. Association for Computational Linguistics. http://www.aclweb.org/anthology/P10-4014.