Thesis for the Degree of Master of Science

Urdu Morphology, Orthography and

Lexicon Extraction

Muhammad Humayoun

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

October 2006
Urdu Morphology, Orthography and Lexicon Extraction
Muhammad Humayoun

© Muhammad Humayoun, October 2006
Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Telephone + 46 (0)31-772 1000

Printed at Chalmers, Göteborg, Sweden, 2006
Abstract

This Thesis work describes the implementation of Urdu language morphology in Functional Morphology. Functional Morphology (FM) is a domain embedded language for implementing natural language morphology, developed by Markus Forsberg and Aarne Ranta at Chalmers & Gothenburg University (Forsberg & Ranta, 2004). Functional Morphology (FM) is written in Haskell, which is a functional programming language. The productivity of this toolkit has been proven by successful implementations of the morphologies for Swedish, Italian, Russian, Spanish and Latin. In this thesis work, it has been shown that this toolkit is equally useful on south Asian languages such as Urdu/Hindi.

Functional Morphology is based on a very simple idea: dealing grammars as software libraries. Therefore this implementation of Urdu morphology could be reused in applications such as "intelligent" search of keywords, infrastructure for syntax & semantics and language training.

Keywords: Urdu, Urdu Morphology, Urdu Orthography, Hindi, Lexicon Extraction, Functional Morphology, Functional Programming, Natural Language Technology.
Acknowledgments:

I would like to thank my supervisor and examiner Professor Aarne Ranta for his invaluable guidance and support. Aarne is always been very helpful. He gave me lot of freedom and encouraged me on every step during this thesis work. It might be difficult to accomplish this work without his guidance.

Further I would like to thank Harald Hammarström and Markus Forsberg for their continuous feedback and suggestions during this thesis work.

I also would like to thank Björn Bringert for helping me troubleshoot the software for interfacing Functional Morphology using Java.

Thanks to Ken Beesly for giving me some general ideas about Urdu transliteration during my internship at Xerox, France.
# Table of Contents

1 INTRODUCTION ........................................................................................................... 6

1.1 Contribution ............................................................................................................. 8

2 FUNCTIONAL MORPHOLOGY .................................................................................. 10

2.1 Haskell .................................................................................................................. 10

2.2 Morphology ........................................................................................................ 10

2.3 Functional Morphology ...................................................................................... 11

2.4 Overview of the System
   2.4.1 Language Independent part of the system .................................................. 12
   2.4.2 Language dependent part of the system ................................................... 14

3 ANALYSIS OF URDU ............................................................................................... 15

3.1 Urdu language ...................................................................................................... 15

3.2 Brief History of Urdu .......................................................................................... 16

3.3 Urdu Orthography
   3.3.1 Urdu Character set ......................................................................................... 16
      3.3.1.1 The Alphabet ....................................................................................... 18
      3.3.1.2 The Vowels (Aərɑb / Hərkɑ) ................................................................. 21
      3.3.1.3 Other Symbols ...................................................................................... 22
         3.3.1.3.1 Numbers ....................................................................................... 23
         3.3.1.3.2 Punctuation: ............................................................................... 23
         3.3.1.3.3 Honorifics: .................................................................................. 23
         3.3.1.3.4 Other Symbols: .......................................................................... 24

3.4 The Transliteration .............................................................................................. 24

3.5 Tools developed
   3.5.1 Urdu Transliterator .................................................................................... 26
   3.5.2 Urdu Extractor ............................................................................................. 28
   3.5.3 Urdu Keyboard Input Method .................................................................... 29
   3.5.4 The Main GUI application ......................................................................... 29

4 URDU MORPHOLOGY AND ITS IMPLEMENTATION IN HASKELL ............ 31

4.1 Nouns ................................................................................................................... 31
   4.1.1 The overview of Case system: ................................................................. 33
      4.1.1.1 Nominative ....................................................................................... 33
      4.1.1.2 Ergative ............................................................................................ 34
      4.1.1.3 Accusative ....................................................................................... 35
      4.1.1.4 Dative ............................................................................................. 35
      4.1.1.5 Instrumental..................................................................................... 36
4.1.1.6 Genitive .................................................................................................................37
4.1.1.7 Locative ..................................................................................................................37
4.1.1.8 Vocative ..................................................................................................................38
4.1.1.9 Oblique ...................................................................................................................39
4.1.2 Implementation of Noun ..........................................................................................40

4.2 Adjectives ....................................................................................................................55

4.3 Verbs .............................................................................................................................58
4.3.1 Verb categories: .........................................................................................................58
4.3.2 Group 1: .....................................................................................................................62
4.3.3 Group 2: .....................................................................................................................63
  4.3.3.1 Group 2.1: .............................................................................................................64
  4.3.3.2 Group 2.2: .............................................................................................................65
  4.3.3.3 Group 2.3: .............................................................................................................66
4.3.4 Group 3: .....................................................................................................................66
4.3.5 Group 4: .....................................................................................................................67
4.3.6 Verb conjugations: ................................................................................................. 67
  4.3.6.1 Single-word-analysis of a verb ........................................................................... 69
  4.3.6.2 Combination-analysis of a verb ........................................................................ 70

4.4 Adverbs ..........................................................................................................................72

4.5 The Closed classes .......................................................................................................72
4.5.1 Pronouns ...................................................................................................................72
  4.5.1.1 Personal pronouns: ..............................................................................................72
  4.5.1.2 Demonstrative Pronoun: ....................................................................................73
  4.5.1.3 Reflexive Pronoun: ...........................................................................................73
  4.5.1.4 Interrogative pronoun: .......................................................................................74
  4.5.1.5 Indefinite Pronoun .............................................................................................74
  4.5.1.6 Repeated Pronoun ............................................................................................ 75
  4.5.1.7 Relative pronoun ...............................................................................................75
4.5.2 PostPositions, Particles and Numerals .................................................................. 76

5 THE LEXICON .................................................................................................................77
5.1 The Extraction of Lexicon ......................................................................................... 77

6 RELATED WORK .............................................................................................................80
6.1 Morphology and Syntax Treatments .........................................................................80
  6.1.1 The Parallel Grammar Project (ParGram) ..............................................................80
  6.1.2 The CRL Language Resources Project ............................................................... 80
  6.1.3 The EMILLE Project ..............................................................................................81
  6.1.4 An Urdu ATN morphological parser ................................................................... 81
6.2 Transliteration Systems ............................................................................................ 81
  6.2.1 Hindi Urdu Machine Transliteration System (HUMTS) .........................................81
  6.2.2 Hindi to Urdu Transliterator ................................................................................81
6.3 Electronic Lexicons ....................................................................................................82

7 CONCLUSION AND FUTURE WORK ...........................................................................83
Chapter 1

1 Introduction

This thesis work presents an implementation of Urdu morphology in the functional language Haskell by using a toolkit for writing morphologies - Functional Morphology (FM henceforth). During my education at Computer Science & Engineering Department at Chalmers, I have studied a course named “Natural Language Technology”. This course developed my interest in this field. Specifically, I became familiar with FM and I have seen the already built resources for some natural languages. This inspired me to pursue this thesis work.

This thesis report is divided into seven chapters. The first chapter describes the introduction and an overview of the work which is done in this thesis. In the second chapter, a detailed analysis of Urdu orthography has been presented. Further more a transliteration scheme and useful GUI tools have been provided to accommodate complex Persho-Arabic script of Urdu. Third chapter describes the FM in detail. Forth Chapter describes Urdu grammar and its implementation with respect to morphology. A wide-coverage lexicon is a key part of any morphological implementation. The fifth chapter is about the lexicon of Urdu. It also describes the methods and tools that were used for the extraction of a lexicon. In the sixth chapter, we described some of the related work in the related field. In the seventh chapter, we have shown the results and pointed out some possible future work. Appendix A is a manual for lexicographers so that words can be added into the lexicon. Appendix B contains a complete set of conjugation for verbs that have been implemented in this work; while Appendix C provides the inflection tables for personal pronouns and relative pronouns.

Urdu/Hindi is the second most widely spoken languages in the world [(Rahman 2004, page 2), (Grimes 2000)]. Having a complex script and grammar, Urdu imposes new challenges in the field of Natural language processing, Machine translation and Content analysis. Unfortunately limited open source research has been done in this field till today as compared to the number of speakers of this language. However, the growing use of Unicode characters and internationalization of softwares has not only shown greater research needs for Urdu language but also provide feasible opportunities and ways to proceed further in this field for a researcher.

Today the state of the art technology to write morphologies is to use special-purpose morphology languages based on finite-state technology. The most well-known among others is XFST (Xerox Finite State Tool) which is based on regular expressions. Xerox Finite State Tools are mathematically elegant, flexible and modifiable. In terms of efficiency they are very fast on expense of large networks. Instead of writing conventional code, one needs to write the grammatical rules directly. A runtime code is applied by the software to the linguistic input for generating a required output (morphology, tokenization, part of speech tagging etc). However, this approach points following questions:

- Does the direct coding allow implementing the linguistic abstraction adequately?
- Is the linguistic model based on regular expressions extensible and reusable? And
- How can it be integrated into applications such as localization, API and user Interfaces?
One non-technical but very important issue related to these tools is that these are commercial products.

On the other hand FM is a toolkit/domain embedded language for defining natural language morphologies written in Haskell. Haskell is a powerful, modern, purely functional programming language, having support of higher-order functions, type classes, polymorphism and a strong type system. Due to the strong type system of Haskell, FM treats the part of speech (categories/word classes) as data types enabling to define a complete type safe linguistic model of a language. It is very easy to add new word classes later (extensible) with very minor efforts. Furthermore morphology written in FM is completely reusable. An example of reusability could be the use of FM in Grammatical Framework (GF henceforth). GF is a special-purpose programming language for grammars which is based on type theory. GF is a functional programming language. A morphology implementation written in FM let the GF extend it seamlessly from morphology to syntax and semantics.

The Overall picture of the Urdu Morphology is shown in the following diagram:

![Urdu Script (Unicode enabled Urdu)](image1)

![Transliteration](image2)

![ASCII / Roman Urdu](image3)

![Language Dependant (Urdu)](image4)

![Morphology (Types, Rules, Lexicon)](image5)

![FM API](image6)

![Language Independent Module](image7)

![Dictionary format](image8)

![Analyzer](image9)

![Exporter](image10)

![Synthesizer](image11)

**Functional Morphology Toolkit**

Fig 1.1
1.1 Contribution

Implementing morphology for Urdu also requires dealing with orthography. This work provides following scientific contribution from our side:

1) An implementation of Urdu morphology as an open source software API having:
   i) A type system that covers language abstraction of Urdu completely.
   ii) A complete inflection engine that covers word and paradigm morphological grammatical rules for Urdu completely.
   iii) Rules for automatic lexicon extraction using extract tool (Forsberg, Hammarström & Ranta, 2006)
   iv) A lexicon of 4131 words and 496976 word forms.
   v) A manual for users/lexicographers to add new words into the lexicon
2) A Unicode Infrastructure for the Urdu morphology API to accommodate complex Perso-Arabic script of Urdu.
3) A demo application and useful GUI tools to provide Urdu morphological analysis both in English and Urdu (Roman transliteration as well as Urdu script).

In this thesis work, we mainly focus on an implementation of morphological or inflectional aspects of Urdu grammar.
We provide following two kind of morphological analysis.

1) Single-word-analysis
2) Combination-analysis

As evident from the name, single-word-analysis means the morphological analysis of all parts of a sentence separated from each other by a space character.

In Urdu word classes (nouns, verbs etc), postpositions (clitics) and auxiliaries are used frequently to explain different cases, tenses, aspects and moods. However in most of the cases, these postpositions and auxiliaries appear in front (sometimes behind) of the word as a separate word. Therefore to know about the case of a noun or the tense of a verb, it is important to analyze words by combining them with such appeared postpositions and auxiliaries. In combination-analysis, morphological analysis of such combinations is displayed. However this analysis is only applicable to the following compositions:

- The noun-clitic compositions:
  Noun + one clitic, combined by a minus (-) sign, to analyze case. e.g. (kəmre-ka, کمرے-کا, room’s), (kəmre-se, کمرے-سے, from room) etc

- The verb-auxiliaries compositions:
Verb + one/two auxiliaries, combined by minus (-) sign, to analyze aspect, mood and tense. e.g. (قُرَا- ئِنَّا, қани, came), (أَرْكِي- ئِنَّا, қانية, was coming) etc

- The *adjective-clitic* compositions:

  Adjective + one/two clitics, combined by minus (-) sign, to analyze the degree of an adjective. e.g. (بَأْسَكَعُ- أَطْطُثُ,更好), (سَبَأَتْ- أَطْطُثُ, بَيْنَتْ- أَطْطُثُ, best) etc

To understand the above mentioned notions of analysis, consider the following sentences:

1) **وَهُوَ آتَارِیسِ غَا**

<table>
<thead>
<tr>
<th>wo</th>
<th>q̱a</th>
<th>ṟehe</th>
<th>ga</th>
</tr>
</thead>
<tbody>
<tr>
<td>He (Pron)</td>
<td>come (Imperf)</td>
<td>remain (verb-aux)</td>
<td>will (Fut. verb-aux)</td>
</tr>
</tbody>
</table>

He will keep coming.

Single-word-analysis: [wo: Pron], [q̱a: Imperf], [ṟehe: verb-aux], [ga: (Fut. verb-aux)]

Combination-analysis:

- [wo: Pron], [q̱a: Imperf], [ṟehe-ga: verb-aux Fut.]
- [wo: Pron], [q̱a-ṟehe-ga: Fut. Imperf Cont]

2) **عَلَیٰ كِیٰ گَا کِتَابُ**

<table>
<thead>
<tr>
<th>Ali ki</th>
<th>ḵṯab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ali (Noun-Gen)</td>
<td>book (Noun-Nom)</td>
</tr>
<tr>
<td>Ali's book</td>
<td></td>
</tr>
</tbody>
</table>

Single-word-analysis: [Ali: Noun-Nom], [ki: Postposition], [ḵṯab : Noun-Nom]

Combination-analysis: [Ali-ki: Noun-Gen], [ḵṯab : Noun-Nom]

However we believe that handling combination-analysis beyond this should be treated at syntax level as it could be treated more elegantly over there in most of cases.
Chapter 2

2 Functional Morphology

In the first section, a very brief introduction of Haskell and morphology is given. Further in this chapter, we have discussed about FM, the overview of this API and its available tools.

2.1 Haskell

Haskell is a functional programming language. It is very high-level, expressive, concise and a type safe language.

As described on official page of Haskell (Haskell Introduction):
“It is a polymorphically typed, lazy, purely functional language, quite different from most other programming languages. The language is named for Haskell Brooks Curry, whose work in mathematical logic serves as a foundation for functional languages. Haskell is based on lambda calculus, hence the lambda we use as a logo.”

Another place at the same page, it is written:

“Haskell is a modern, standard, non-strict, purely-functional programming language. It provides all the features sketched above, including polymorphic typing, lazy evaluation and higher-order functions. It also has an innovative type system which supports a systematic form of overloading and a module system.
It is specifically designed to handle a wide range of applications, from numerical through to symbolic. To this end, Haskell has an expressive syntax, and a rich variety of built-in data types, including arbitrary-precision integers and rationals, as well as the more conventional integer, floating-point and boolean types."

2.2 Morphology

Morphology is a branch of linguistics that studies the structure of words and their different dictionary forms. A meaningful linguistic unit consisting of a word that cannot be divided into smaller meaningful parts is called morpheme. It could be a complete word such as man, or a word element, such as “ed” and “walk” in walked.
Words can be divided into the lexical categories (word class/part of speech). In morphology, words are divided into their lexical categories with the help of their inflection and by the position in which they (the words) are allowed to occur (rules). Nouns, verbs, adjectives, adverbs are examples of some common lexical categories among most of the languages.

To show a brief example on how to decide the lexical category of a word, suppose a noun “ئرکا، یسج (boy)”. In Urdu grammar, nouns are normally inflected in number (singular, plural) and case (nominative, genitive, and accusative etc). So any word inflected in number and case,
could be considered noun. Therefore due to the inflection of “ləɽkɑ, kDyF (boy)” into “ləɽkɑ, kDyF (boy)”, “ləɽke, kDyF (boys)”, “ləɽke-ka, kDyFka, (boy’s)” etc, we could conclude “ləɽkɑ, kDyF (boy)” as a noun.

Some examples of the usability of morphology could be machine translation, information retrieval, software localization and language education. A more specific case of language education for which computational morphology is useful is CALL (computer assisted language learning).

2.3 Functional Morphology

FM is a toolkit for morphology development in a functional programming language, Haskell, developed by Markus Forsberg and Aarne Ranta (Forsberg & Ranta, 2004). It is based on an idea of using the high expressiveness provided by functional languages to define morphology. The use of Haskell gives access to powerful programming constructs and high level of abstraction, which is very useful to capture the generalization of a natural language.

FM is influenced by Gérard Huet's work (Huet, 2000) and his Zen Tool kit (Huet, 2002). Gérard Huet has implemented the Sanskrit morphology in a functional language CAML. Further he has generalized the ideas used for Sanskrit to a toolkit for computational linguistics named Zen. In a similar way, the FM toolkit is a successful experiment of how the morphology can be implemented in Haskell by using it as a host language while FM acts as a domain specific embedded language. The productivity and reliability of Haskell for this task has been proved by successful implementations of the morphologies for Swedish, Italian, Russian, Spanish and Latin into Haskell (FM, 2004).

The words can be searched at the speed up to 2k-50k words per second (depending on how much compound analysis is involved) by the analyzer (Forsberg & Ranta, 2004, Page 10). The analyzer is a key component in a morphology system to analyze a word into its lemma and its grammatical description. Decorated tries is currently used instead of transducers for analysis in FM which compiles very fast because Kleene's star is disallowed within a word description.

FM library can also be tagged as a morphological part of Grammatical Framework (GF). GF is special-purpose programming language for grammars being designed at the Language Technology Group, Chalmers & Gothenburg University. GF is also a functional programming language, based on type theory. Although morphological implementation of a language can be written directly in GF, but FM provides more control, freedom and functionality for defining morphology due to the powerful programming constructs of Haskell. Further a morphology implementation written in FM let GF extend it seamlessly from morphology to syntax and semantics. Therefore by using FM and GF, it becomes very easy to separate morphology from syntax and semantics to let a linguistic developer concentrate on each part better.
My scientific contribution to this project is an implementation of Urdu language morphology. This Thesis work can also be seen as an experiment if FM can adequately capture the abstraction of Urdu. To know more about FM, one can look at the papers listed at (FM, 2004).

2.4 Overview of the System

FM defines the paradigms (inflection tables) as finite functions over enumerable, finite, algebraic data types that describe the parameters of the paradigm. FM consists of two parts as shown in figure 2.1:

- Language independent part
- Language dependent part

2.4.1 Language Independent part of the system

Language independent part of system consists of following components:

1. Infrastructure for Dictionary compilation
2. Runtime applications (Analyzer, Synthesizer)
3. Data Export Utility (Translator)

Translator can export the lexicon in the following different formats for compatibility:
- Full form lexicon listing all word forms with their analysis, alphabetically!
- Inflection tables in Postscript format
- GF grammar source code
- An XML representation of morphological lexicon
- XFST source code: for simple, non-cyclic transducers in Xerox notation
- LEXC source code: for LEXC format, a version of XFST that is optimized for morphological descriptions
- SQL database: described by SQL source code
- Decorated tries: an Analyzer for the morphology as decorated trie

FM consists of three main type classes: Param, Dict and Language. These type classes enable code reuse and provide generic algorithms for analysis, synthesis and code generation. We will only discuss General.hs here. For more details one should check the source code and FM home page (FM, 2004).
General.hs:
It provides language-independent morphology data types and operations. A morphology implementing developer should take a look on General.hs because most of the useful utility functions (string manipulation, exceptions, representing non-existing forms for an inflections etc) are defined in it. Here I will only explain some of them.

In Urdu, noun is described (inflected) in number and case. Suppose a word (کمرہ, Room). Some of its forms could be:

Nominative-Singular: (کمرہ, Room)
Nominative-Plural: (کمرہے, Rooms) and so on.

Where nominative-singular and nominative-plural could belong to a type Noun. This assignment is done by using a Table data structure defined in General.hs in the following way:

type Table a = [(a, Str)]

Where type Str is defined as a list of strings, while “a” is a type variable, for the moment representing nouns and straight brackets show that the Table “a” is a list of values.

So the inflection of (کمرہ, Room) is represented as:

[(Nominative-Singular, کمرہ), (Nominative-Plural, کمرہے), .............]

Parameter types like number and case are language dependent part of FM and to be able to use the defined functions in General.hs, they should be the valid instances of Param class. However It is important to note the role of functions in FM, as one of the most important ideas behind FM is to use functions instead of explicit tables (i.e. Param a => [(a, Str)]) for the paradigm descriptions. This idea enables a developer to build up inflection functions by a set of auxiliary functions (linguistic abstractions) and more directly, the pattern matching engine can be used to capture common cases. As the data types are enumerable finite (through Param), the inflection table can be easily generated by enumerating the function's inflectional type; and by doing this, a completeness check is also performed automatically, i.e. that all cases are defined.

The following four string functions defined in General.hs are very useful and should be mentioned here:

*tk* drops n final letters from a string and returns the remaining string
*dp* return n final letters from the string
*nonExist*: It can use to represent the missing forms in a certain inflection table
*mkStrWords*: If a form is represented by more then one string variant, this function can be used to encode such situations. For example the in Urdu noun (کمرہ, Room), the Genitive case can be encoded with this function as it has two forms- Genitive-Singular: (کمرہ-کا, Room’s) and (کمرہ-کی, Room’s)
2.4.2 Language dependent part of the system

This is the part that a morphology implementer has to provide. The implementation then will be a new library on the top of the language independent part of FM. Language dependent part consists of following modules:

- A type system
- An Inflection engine
- A lexicon

All word classes and the parameters belonging to them are defined as algebraic data types in the type system which is represented in TypesUrdu.hs for this implementation. An Inflection engine defines all possible inflection tables (paradigms) for all word classes. It is represented in RulesUrdu.hs. A lexicon provides a list of all words in the target language with their paradigms. The words belonging to closed classes are added in DictUrdu.hs and the words belonging to open classes added in urdu.lexicon. In both cases the name of an interface function is provided to identify to which group a certain word belongs. BuildUrdu.hs behaves like a coordinator between type system (TypesUrdu.hs), Inflection engine (RulesUrdu.hs) and the lexicon (urdu.lexicon, DictUrdu.hs). Here the interface functions are defined for all word classes by using the classification types and the inflection functions. The implementation of this part for Urdu is discussed in chapter 4 in detail.
Chapter 3

3 Analysis of Urdu

3.1 Urdu language

Urdu is an Indo-Aryan language widely spoken in Pakistan, the northern parts of India and in Jammu & Kashmir. The language family tree of Urdu is described as: Indo-European→Indo-Iranian→Indo-Aryan→Urdu.

In discussion about Urdu one often come across the terms Urdu, Hindi, Hindustani and Hindi-Urdu. Urdu and Hindi are similar in their grammatical structure and semantics but are different in script, phonology and some of the vocabulary. Urdu has a strong Perso-Arabic influence in its vocabulary and is written in a Perso-Arabic script from right to left; whereas Hindi has a strong influence of Sanskrit and the other native languages of India and is written in Devanagari script from left to right. There can also be found several major differences between Urdu and Hindi at phonological level (Naim, 1999, preface: page iii); e.g. having multiple letters to represent the same phonetic sound etc. We will not discuss them further as they are out of the scope of this thesis work.

Hindustani refers to a language without any influence of Sanskrit, Persian or Arabic in terms of borrowing vocabulary. Hindi-Urdu is normally used to study what is common between Urdu and Hindi.

Despite the differences discussed above, both languages share a huge amount of vocabulary and same grammatical rules. According to (Rai, 2000), “one man’s Hindi is another man’s Urdu”. In this thesis work we will stick with the modern Urdu widely spoken in Pakistan and practiced in literature.

Urdu is the one of the official languages of Pakistan, India (Andhra Pradesh, Delhi and Uttar Pradesh) and Jammu & Kashmir. It is also spoken world-wide due to the south Asian Diaspora (big population in Middle East, USA, UK, Norway and Canada etc).

According to (Rahman 2004, page 2) and (Grimes 2000), Urdu-Hindi is the second most spoken language in the world after Chinese.

The following table gives an idea of its size in numbers which is taken from (Rahman 2004, page 2).

<table>
<thead>
<tr>
<th></th>
<th>Mother Tongue Speakers</th>
<th>Second Language Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hindi</td>
<td>366,000,000</td>
<td>487,000,000</td>
</tr>
<tr>
<td>Urdu</td>
<td>60,290,000</td>
<td>104,000,000</td>
</tr>
<tr>
<td>Total</td>
<td>426,290,000</td>
<td>591,000,000</td>
</tr>
<tr>
<td>Grand Total: Mother tongue + second language speakers of Urdu-Hindi = 1,017,290,000.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2 Brief History of Urdu

In 712 CE, Islam came to the South Asia with the conquest of the some parts of India (which is today Sindh and Punjab, the provinces of Pakistan) by an Arab Muslim General Muhammad bin Qasim, who came to rescue Muslim women and children from Raja Dahir, the ruler of Sindh. The conquest of Sindh and Punjab started an Islamic era in South Asia. After that, there were many Muslim rulers, coming from central Asia (Turks, Mongols, Iranians, Afghans and Mughals; who itself has hybrid ethnicity), conquered the most of the areas of South Asia (currently Pakistan, India, Bangladesh and Nepal), Afghanistan and some parts of Iran, establishing the first Muslim empire in South Asia which last from 13th century to 1857. This mixed environment in South Asia resulted in a hybrid language of Arabic, Pashto, Turkish, Persian and native languages of region, which is today eventually, be known as Urdu. Urdu word itself is a Turkish word meaning “tent” or "army". Urdu came into existence due to the interaction of Muslim soldiers and the native speakers of the region. Urdu soon was used as an official language in Mughal era and gained a distinction as an important language of the region due to its wider understanding among different ethnic groups. Despite being a younger language of the region, an enormous amount of literature and poetry can be found in Urdu today.

3.3 Urdu Orthography

Urdu has a derivative Persian script which is itself a derivative of Arabic script with some addition of new letters under influence of the native languages of the region. Urdu is mostly written in Nastaleeq script which is a cursive, context-sensitive and complex system of writing for Perso-Arabic scripts. Urdu is written from right to left.

3.3.1 Urdu Character set

The Urdu alphabet has been standardized by National Language Authority (NLA), which is a regulating body for Urdu Standards in Pakistan. The standard for Urdu alphabet is known as Urdu Zabta Takhti (UZT 1.01 Afzal & Sarmad, 2001, page 2) means Urdu standard character set. The whole table can be viewed below which is taken from a Master’s Thesis (Malik, 2006, chapter 2, page 5) with some small modifications mentioned after the table.
<table>
<thead>
<tr>
<th>Abbreviations:</th>
<th>Legend</th>
<th>Box Explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sp: Space, Cr: Currency, Dc: Decimal, Dv: Division, HS: Hard Space, US: Under Score, Da: Dash, ▼: Code Plate Switching</td>
<td><img src="image" alt="Control Area (Not to be used)" /> <img src="image" alt="Reserved Area (for future use)" /> <img src="image" alt="Vender Area" /></td>
<td>▼: IPA 0628</td>
</tr>
</tbody>
</table>

Slight updating is done at two places in above table according to Arabic Unicode standard. Modifications:
(Column: 4, Row: 7); Unicode Provided
(Column: 4, Row: 8); Unicode Provided

Abbreviations:

Legend
Control Area (Not to be used)  Reserved Area (for future use)  Vender Area

Box Explanation:
▼: IPA 0628
At some places it seems that a slightly different Urdu letter is provided as compared to (Malik, 2006, chapter 2, page 5). This difference is observed only due to the use of different font here in the table above (Nafees Web Naskh). Such places are following:
Letters at (Column: 2, Row: 7), (Column: 2, Row: 2), (Column: 3, Row: B) and (Column: A Row: E).

Following are the logical sections of Urdu Zabta Takhti (UZT 1.01 Afzal & Sarmad, 2001).

- Alphabet (80 – 122)
- Ḥerkaṭ/ ʾAerab / diacritics (66 – 79, 123 – 126)
- Other letters
  - Punctuation and arithmetic symbols (32 – 47, 58 – 65)
  - Digits (48 – 57)
  - Special symbols (160 – 176, 192 – 199)
  - Miscellaneous
    - Control characters (0 – 31, 127)
    - Reserved control space (128 – 159, 255)
    - Vendor area (208 – 239)
    - Toggle character (254)

There is a slight disagreement about the number of letters in Urdu alphabet. According to (UZT 1.01 Afzal & Sarmad, 2001, page 2), Urdu consist of 57 letters whereas according to (Siddiqi 1971, page 207), there are fifty two letters. However the disagreement appears due to some of the letters which only present in very small number of words and the difference of opinion about the vowels if they should be considered as letters or not. This discussion is not very important since we can locate all the letters with their IPA and Unicode from table 3.1.

Urdu Character set can be divided into three parts

1. The Alphabet
2. The Vowels (Ḥerkaṭ/ ʾAerab)
3. Other Symbols

### 3.3.1.1 The Alphabet

The Alphabet can be divided into two parts

- The Non Aspirated letters
- The Aspirated letters

The following forty one letters are non-aspirated, which are represented by a single character and produce a single voice:
### Non-aspirated letters

<table>
<thead>
<tr>
<th>Letter</th>
<th>Name of letter</th>
<th>Pronunciation in the IPA (International Phonetic Alphabet)</th>
<th>Unicode</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ب</td>
<td>be</td>
<td>[b]</td>
<td>0628</td>
<td>b</td>
</tr>
<tr>
<td>پ</td>
<td>pe</td>
<td>[p]</td>
<td>067E</td>
<td>p</td>
</tr>
<tr>
<td>ت</td>
<td>ūt</td>
<td>Dental [t̪] Close to French t as in trios.</td>
<td>062A</td>
<td>t</td>
</tr>
<tr>
<td>ث</td>
<td>ūth</td>
<td>retroflex [ʈ]</td>
<td>0679</td>
<td>T</td>
</tr>
<tr>
<td>س</td>
<td>se</td>
<td>[s] Close to English s</td>
<td>062B</td>
<td>C</td>
</tr>
<tr>
<td>ج</td>
<td>dʒim</td>
<td>[dʒ] Same as English j</td>
<td>062C</td>
<td>j</td>
</tr>
<tr>
<td>چ</td>
<td>ūtʃe</td>
<td>[ʈʃ] Same as English ch, not like Scottish ch</td>
<td>0686</td>
<td>e</td>
</tr>
<tr>
<td>ح</td>
<td>bəɾi he</td>
<td>[h] voiceless h</td>
<td>062D</td>
<td>H</td>
</tr>
<tr>
<td>خ</td>
<td>xe</td>
<td>[x]</td>
<td>062E</td>
<td>K</td>
</tr>
<tr>
<td>د</td>
<td>ɡal</td>
<td>dental [ɡ]</td>
<td>062F</td>
<td>d</td>
</tr>
<tr>
<td>ذ</td>
<td>ɡal</td>
<td>retroflex [ɡ]</td>
<td>0688</td>
<td>D</td>
</tr>
<tr>
<td>ر</td>
<td>re</td>
<td>dental [r]</td>
<td>0631</td>
<td>r</td>
</tr>
<tr>
<td>ئ</td>
<td>re</td>
<td>retroflex [r̪]</td>
<td>0691</td>
<td>R</td>
</tr>
<tr>
<td>ز</td>
<td>ze</td>
<td>[z]</td>
<td>0632</td>
<td>z</td>
</tr>
<tr>
<td>ص</td>
<td>ze</td>
<td>[ʒ]</td>
<td>0634</td>
<td>X</td>
</tr>
<tr>
<td>ض</td>
<td>sin</td>
<td>[s]</td>
<td>0633</td>
<td>s</td>
</tr>
<tr>
<td>ض</td>
<td>zu'ad</td>
<td>[s]</td>
<td>0635</td>
<td>S</td>
</tr>
<tr>
<td>ط</td>
<td>to'e</td>
<td>[t̪]</td>
<td>0637</td>
<td>t</td>
</tr>
<tr>
<td>ئ</td>
<td>zu'ad</td>
<td>[z]</td>
<td>0638</td>
<td>ŋ</td>
</tr>
<tr>
<td>ع</td>
<td>'æn</td>
<td>[ɑ] after a consonant; otherwise [ʔ], [ə], or silent.</td>
<td>0639</td>
<td>e</td>
</tr>
</tbody>
</table>
Table 3.2

<table>
<thead>
<tr>
<th>Letter</th>
<th>Name of letter</th>
<th>Pronunciation in the IPA (International Phonetic Alphabets)</th>
<th>Unicode</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>غ</td>
<td>yaeen</td>
<td>[ɣ]</td>
<td>063A</td>
<td>G</td>
</tr>
<tr>
<td>ف</td>
<td>fe</td>
<td>[f]</td>
<td>0641</td>
<td>f</td>
</tr>
<tr>
<td>ق</td>
<td>qaf</td>
<td>[q]</td>
<td>0642</td>
<td>q</td>
</tr>
<tr>
<td>ك</td>
<td>kaf</td>
<td>[k]</td>
<td>069A</td>
<td>k</td>
</tr>
<tr>
<td>ك</td>
<td>gaf</td>
<td>[g]</td>
<td>06AF</td>
<td>g</td>
</tr>
<tr>
<td>ل</td>
<td>lam</td>
<td>[l]</td>
<td>0644</td>
<td>l</td>
</tr>
<tr>
<td>م</td>
<td>mim</td>
<td>[m]</td>
<td>0645</td>
<td>m</td>
</tr>
<tr>
<td>ن</td>
<td>nun</td>
<td>[n] or a nasal vowel</td>
<td>0646</td>
<td>n</td>
</tr>
<tr>
<td>و</td>
<td>va’o</td>
<td>[v, u, o, ow]</td>
<td>0648</td>
<td>w</td>
</tr>
<tr>
<td>ه</td>
<td>tʃho’ti he</td>
<td>[ɑ] at the end of a word, otherwise [h]</td>
<td>06C1</td>
<td>h</td>
</tr>
<tr>
<td>ه</td>
<td>do-tʃeʃmi he</td>
<td>[ʰ], Indicates that the preceding consonant is aspirated (p, t, ch, k) or murmured (b, d, j, g).</td>
<td>06BE</td>
<td>ʰ</td>
</tr>
<tr>
<td>ة</td>
<td>te dairwi</td>
<td>[t]</td>
<td>06C3</td>
<td>t</td>
</tr>
<tr>
<td>ی</td>
<td>tʃho’ti ye</td>
<td>[j, l, e, ɛ]</td>
<td>06CC</td>
<td>y</td>
</tr>
<tr>
<td>ہ</td>
<td>bəɾi ye</td>
<td>[ɛ]</td>
<td>06D2</td>
<td>E</td>
</tr>
<tr>
<td>ء</td>
<td>əlɪf hamza</td>
<td></td>
<td>0623</td>
<td>a^</td>
</tr>
<tr>
<td>ن</td>
<td>va’o hamza</td>
<td></td>
<td>0624</td>
<td>w^</td>
</tr>
<tr>
<td>ی</td>
<td>tʃho’ti ye</td>
<td></td>
<td>0626</td>
<td>y^</td>
</tr>
</tbody>
</table>

Table 3.3

Vowel Non-aspirated letters

<table>
<thead>
<tr>
<th>Letter</th>
<th>Name of letter</th>
<th>Pronunciation in the IPA (International Phonetic Alphabets)</th>
<th>Unicode</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>آ</td>
<td>əlɪf medda</td>
<td>[a]</td>
<td>0622</td>
<td>A</td>
</tr>
<tr>
<td>ا</td>
<td>əlɪf</td>
<td>[æ, a, ʔ] after a consonant; silent when initial. Close to an English long a as in mask</td>
<td>0627</td>
<td>a</td>
</tr>
<tr>
<td>ن</td>
<td>nun ɣuna</td>
<td>[n]</td>
<td>06BA</td>
<td>N</td>
</tr>
<tr>
<td>ء</td>
<td>həmza</td>
<td>[ʔ] or silent</td>
<td>0621</td>
<td>&amp;</td>
</tr>
</tbody>
</table>

Table 3.3
Following twelve letters are aspirated, compound of two characters each:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Name of letter</th>
<th>Pronunciation in the IPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bʰe</td>
<td>[bʰ] murmured voice</td>
</tr>
<tr>
<td>2</td>
<td>pʰe</td>
<td>[pʰ] aspirated voice</td>
</tr>
<tr>
<td>3</td>
<td>tʰe (plosive)</td>
<td>[tʰ] aspirated voice</td>
</tr>
<tr>
<td>4</td>
<td>tʰe</td>
<td>[tʰ] aspirated voice</td>
</tr>
<tr>
<td>5</td>
<td>dʒʰe</td>
<td>[dʒʰ] aspirated voice</td>
</tr>
<tr>
<td>6</td>
<td>tʃʰe</td>
<td>[tʃʰ] aspirated voice</td>
</tr>
<tr>
<td>7</td>
<td>d̪ʱe</td>
<td>[d̪ʰ] murmured voice</td>
</tr>
<tr>
<td>8</td>
<td>d̪ʰe</td>
<td>[d̪ʰ] aspirated voice</td>
</tr>
<tr>
<td>9</td>
<td>rʱe</td>
<td>[rʰ] murmured voice</td>
</tr>
<tr>
<td>10</td>
<td>rʰe</td>
<td>[rʰ] aspirated voice</td>
</tr>
<tr>
<td>11</td>
<td>kʰe</td>
<td>[kʰ] aspirated voice</td>
</tr>
<tr>
<td>12</td>
<td>gʰe</td>
<td>[gʰ] aspirated voice</td>
</tr>
<tr>
<td>13</td>
<td>mə̂n</td>
<td>[mʰ] murmured voice</td>
</tr>
<tr>
<td>14</td>
<td>lə̂n</td>
<td>[lʰ] murmured voice</td>
</tr>
<tr>
<td>15</td>
<td>nə̂n</td>
<td>[nʰ] murmured voice</td>
</tr>
</tbody>
</table>

Table 3.4

3.3.1.2 The Vowels (A'rāb / Hurrāṭ)

The following are the “Familiar” vowels (Hurrāṭ / A'rāb), diacritical marks.
<table>
<thead>
<tr>
<th>No.</th>
<th>Shape</th>
<th>Representation</th>
<th>Description</th>
<th>Code</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>دٍ</td>
<td>peʃ</td>
<td>[ʊ] generally, [ʊ] if ʿalīf (ı) is behind the peʃ and ʿawāʾo (و) is after.</td>
<td>064F</td>
<td>(o)</td>
</tr>
<tr>
<td>5</td>
<td>دٍ</td>
<td>ʿulta-peʃ</td>
<td>[ʊ], Only used in few Arabic loan words</td>
<td>0657</td>
<td>(u)</td>
</tr>
<tr>
<td>6</td>
<td>دٍ</td>
<td>kʰerī-zəbər</td>
<td>[a], Only used in few Arabic loan words</td>
<td>0670</td>
<td>[a]</td>
</tr>
<tr>
<td>7</td>
<td>دٍ</td>
<td>kʰerī-zer</td>
<td>[i], Only used in few Arabic loan words</td>
<td>0656</td>
<td>[i]</td>
</tr>
<tr>
<td>8</td>
<td>دٍ</td>
<td>ḍo-zəbər</td>
<td>[ən]</td>
<td>064B</td>
<td>(A)</td>
</tr>
<tr>
<td>9</td>
<td>دٍ</td>
<td>ḍo-zer</td>
<td>[in]</td>
<td>064D</td>
<td>(I)</td>
</tr>
<tr>
<td>10</td>
<td>دٍ</td>
<td>ḍo-peʃ</td>
<td>[un]</td>
<td>064C</td>
<td>(O)</td>
</tr>
<tr>
<td>11</td>
<td>دٍ</td>
<td>ʃədd or ʃədd</td>
<td>Mark/ emphasize</td>
<td>0651</td>
<td>&quot;</td>
</tr>
<tr>
<td>12</td>
<td>دٍ</td>
<td>ʧəoṭa non</td>
<td>Used to mark the nasalization of a vowel (Malik, 2006, chapter 2, page 14). Not very common.</td>
<td>0658</td>
<td>M</td>
</tr>
<tr>
<td>13</td>
<td>دٍ</td>
<td>dʒəzm</td>
<td>To mark the absence of vowel after base consonant (Platts, 1909), (Malik, 2006) To give a little pause inside a word. Same as Arabic Sukun. On Urdu phonetic keyboard no key is assigned for dʒəzm, instead a key for Arabic dʒəzm (06E1) is assigned, which is incorrect, so for transliteration purpose even if user enters Arabic dʒəzm (06E1), we treat it as Urdu dʒəzm</td>
<td>0652</td>
<td>َ</td>
</tr>
<tr>
<td>14</td>
<td>دٍ</td>
<td>həmza-izafat/ ʧəoṭi  həmza</td>
<td>used to make compound word by connecting two words</td>
<td>0654</td>
<td>^</td>
</tr>
<tr>
<td>15</td>
<td>دٍ</td>
<td>ʧəoṭi to'e</td>
<td></td>
<td>0615</td>
<td>%</td>
</tr>
</tbody>
</table>

Table 3.5

3.3.1.3 Other Symbols

We can divide miscellaneous symbols into following categories:

- Numbers
- Punctuations
- Honorifics
- Other Symbols

### 3.3.1.3.1 Numbers

Although separate Urdu letters exist for representing numbers but Roman letters are also frequently used in literature and Mathematics. Following is the table containing both Urdu and their equivalent Roman letters used for numbers.

<table>
<thead>
<tr>
<th>Roman</th>
<th>Unicode</th>
<th>Transliteration</th>
<th>Urdu</th>
<th>Unicode</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>۰</td>
<td>06F0</td>
<td>0_ur</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>۱</td>
<td>۱</td>
<td>06F1</td>
<td>۱_ur</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>۲</td>
<td>۲</td>
<td>06F2</td>
<td>۲_ur</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>۳</td>
<td>۳</td>
<td>06F3</td>
<td>3_ur</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>۴</td>
<td>۴</td>
<td>06F4</td>
<td>۴_ur</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>۵</td>
<td>۵</td>
<td>06F5</td>
<td>۵_ur</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>۶</td>
<td>۶</td>
<td>06F6</td>
<td>۶_ur</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>۷</td>
<td>۷</td>
<td>06F7</td>
<td>۷_ur</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>۸</td>
<td>۸</td>
<td>06F8</td>
<td>۸_ur</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>۹</td>
<td>۹</td>
<td>06F9</td>
<td>۹_ur</td>
</tr>
</tbody>
</table>

Table 3.6

### 3.3.1.3.2 Punctuation:

<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode</th>
<th>Transliteration</th>
<th>Character</th>
<th>Unicode</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>؟</td>
<td>061F</td>
<td>؟</td>
<td>۰</td>
<td>06D4</td>
<td>NA</td>
</tr>
<tr>
<td>:</td>
<td>061B</td>
<td>NA</td>
<td>۰</td>
<td>003A</td>
<td>NA</td>
</tr>
<tr>
<td>,</td>
<td>060C</td>
<td>NA</td>
<td>۰</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3.7

### 3.3.1.3.3 Honorifics:

These are the special symbol letters, used as an abbreviation to honor some special personalities. These honorifics are rare to find in text and are not useful for morphology.

<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode</th>
<th>Description</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>﷽</td>
<td>FDF2</td>
<td>Allah (A Name of God)</td>
<td>[ALLAH]</td>
</tr>
<tr>
<td>۴</td>
<td>FDFD</td>
<td>In the name of God, the most beneficent, the most merciful</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3.8
3.3.1.3.4 Other Symbols:

Following are some other symbol letters used in Urdu. These are very rare to find in text and are not useful for morphology at all. Therefore transliteration is not provided.

<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode</th>
<th>Serial</th>
<th>Character</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  ب</td>
<td>0614</td>
<td>5  ﮩ</td>
<td>5  ﮩ</td>
<td>0603</td>
</tr>
<tr>
<td>2  ﮩ</td>
<td>060F</td>
<td>6  ﮨ</td>
<td>6  ﮨ</td>
<td>0601</td>
</tr>
<tr>
<td>3  ﮨ</td>
<td>0602</td>
<td>7  ﮩ</td>
<td>7  ﮩ</td>
<td>0653</td>
</tr>
<tr>
<td>4  ﮨ</td>
<td>0603</td>
<td>8  ﮨ</td>
<td>8  ﮨ</td>
<td>FEFB</td>
</tr>
</tbody>
</table>

Table 3.9

3.4 The Transliteration

Transliteration is a very strict, reversible, one to one mapping from one system of writing into another. According to (Beasley, 1998): “The purpose of a Transliteration (sometimes called a "strict transliteration" or "orthographical transliteration") is to write a language in its customary orthography, using the exact same orthographical conventions, but using carefully substituted orthographical symbols. Transliterations are appropriate when one wants to use the traditional orthography (with all its strengths and weaknesses, all its distinctions and ambiguities) but where writing or displaying or storing the original characters is impossible or inconvenient.”

In this MS thesis work, it is decided to store all the work related to Urdu morphology (Type System, grammatical rules, and lexicon) in ASCII characters in order to be able to view and manipulate them easily.

To achieve that, a clear, strict, reversible and one to one string transliteration scheme is defined. It is tried to make this transliteration as phonetic as possible so that even if a user wants to input the Roman version of script, he could do it at the same ease as he could type ASCII English or Urdu script. To achieve this Urdu phonetic keyboard was found very useful.
which is designed by Center for research for Natural Language Processing, Pakistan (CRULP, Urdu phonetic keyboard). However modifications are done where found appropriate.

Furthermore we want the end product to support Unicode Character set. To achieve that four useful programs have been developed (section 3.5) in which strict and non ambiguous conversion has been done between Urdu script, Roman transliteration and Urdu Unicode Character codes. The table (3.2-9) shows mapping of alphabets from Urdu script to Roman.

As noted in the tables (3.2-9), there is no transliteration mapping is provided for aspirated alphabets separately because they can be written using two corresponding roman letters. For example we can write \( \text{ﮨ} \) (ḇe) by using the mapped strings for \((\text{be}, ہ)\) and \((d̪o-tʃəmi he, ہ)\) in transliteration scheme.

\[ \text{ﮨ} \rightarrow \text{b|h} \]

Following are some example words converted from Urdu script to equivalent Roman transliteration.

<table>
<thead>
<tr>
<th>Serial</th>
<th>Urdu</th>
<th>Meaning</th>
<th>Roman</th>
<th>Pronunciation and Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>كتاب</td>
<td>Book</td>
<td>k(i)tab</td>
<td>(kɪt̪ɑb) - vowel ‘(i)’ between ‘k’ and ‘t’ is written as a substitute of zer vowel (.).</td>
</tr>
<tr>
<td>2</td>
<td>كتاب</td>
<td>Book</td>
<td>ktab</td>
<td>(kɪt̪ɑb)</td>
</tr>
<tr>
<td>3</td>
<td>كوشش</td>
<td>Struggle</td>
<td>k(a)wX(i)X</td>
<td>(kʊʃʃ) - vowel zer ‘(a)’ between ‘k’ and ‘w’, vowel zer ‘(i)’ between two ʃi ‘X’</td>
</tr>
<tr>
<td>4</td>
<td>كوشش</td>
<td>Struggle</td>
<td>kwXX</td>
<td>(kʊʃʃ)</td>
</tr>
</tbody>
</table>

Table 3.11

However for the sake of better readability, in this thesis report we write Urdu words in following style:

(Phonetic-transcription, Urdu-script, Roman-transliteration, English meaning) or (Phonetic-transcription, Urdu-script, Roman-transliteration), if English meaning is not important to mention or (Phonetic-transcription, Urdu-script).

Table 3.11 also shows an interesting problem; how word كتاب and كتاب could be treated by the morphological analyser since both represent the same word with the difference of provided orthographic information.

We propose a very simple and straightforward solution:

Lexicon will save all orthographically different words. When user requests for morphological analysis of a word including vowels (Aərab), that word will be searched and if found; it will be displayed with morphological analysis. When user requests for morphological analysis of a word without specifying any vowels (Aərab), all orthographically possible forms of that word
will be generated at runtime and the matches will be shown with their corresponding morphological analysis.

Continuing the same word example, suppose we have a word ‘book, كتاب’ in lexicon, having two different orthographic forms ‘k(i)tab, كتاب’ and ‘ktab, كتاب’.

If a user requests for a morphological analysis of word ‘k(i)tab’, it will be searched and displayed. If a user requests for a morphological analysis of word ‘ktab’, then following forms of word will be generated at runtime:

k(a)t(a)ab, k(i)t(a)ab, k(i)t(o)ab, k(u)t(a)ab, k(i)t(i)ab, k(i)t(i)ab, ..... 

Then these words will be searched separately in the lexicon displaying all the entries that exists in lexicon, resulting the morphological analysis of forms ‘k(i)tab’ and ‘ktab’.

3.5 Tools developed

The following tools have been developed to support Unicode Character set, for transliteration, to extract Urdu text and to let a user type Urdu even if the appropriate fonts and Urdu keyboard is not installed on the system. All the graphic user interfaces for these tools are developed by Java Swing package.

- Urdu Transliterator (Utility for the conversion of Urdu to Roman and Roman to Urdu)
- Urdu Extractor (Utility for Extraction of Urdu text from Web pages or text pages)
- Urdu Keyboard Input Method (Utility to type Urdu without installing Urdu keyboard layout)
- The Main GUI application

3.5.1 Urdu Transliterator

The implementation of transliteration to and from Urdu script is done by using Java and a Transliterator class of ICU4J (ICU4J 3.4, 2006). ICU4J is an implementation of ICU for Java language. ICU is an open source project to implement the software internationalization in software programs. It is developed by IBM Corporation. ICU enables to write language-independent C/C++/Java code that is used on separate, localized resources to get language-specific results. It supports many features, including language-sensitive text, dates, time, numbers, currency, message sorting, and searching.

For Urdu Transliterator, a RuleBasedTransliterator is used which extends com.ibm.icu.text.Transliterator class. Defining rules for translation was rather straight forward in a following manner:

```java
private static final String unicode_to_Roman_rules =
    UrduUnicode.alif_madda   +   "">"   +   UrduRoman.alif_madda   +   ";">"   +
    UrduUnicode.alif         +   "">"   +   UrduRoman.alif     +   "">"   +
    UrduUnicode.bay          +   "">"   +   UrduRoman.bay       +   "">"   +
```
First a public class named "UrduUnicode" is defined containing all the Urdu letters with their
Unicode values:

```java
public class UrduUnicode
{
    public static final char alif_madda='\u0622';
    public static final char alif='\u0627';
    public static final char bay='\u0628';
    public static final char pay='\u067e';
    ...
}
```

In a similar way, a public class named "UrduRoman" is defined consisting of one to one string
mapping for each letter:

```java
public class UrduRoman {
    public static final String alif_madda = "A";
    public static final String alif = "a";
    public static final String bay = "b";
    public static final String pay = "p";
    public static final String tay = "t";
    ...
}
```

Then straight forward transliteration rules were defined in a class named “Transliterator_ur”
as mentioned above.

After creating rules for all letters, an actual transliterator object was created by calling
`createFromRules()` method of `Transliterator` class:

```java
public static final Transliterator unicode_to_roman =
    Transliterator.createFromRules("RomanUrdu-Unicode",
    unicode_to_Roman_rules, 0);
```

In the similar manner, rules for reverse mapping (Roman to Unicode) are defined and another
transliterator object is created:

```java
private static final String roman_to_Unicode_rules =
    UrduRoman.alif_madda + "\" + UrduUnicode.alif_madda + ";" +
    UrduRoman.alif + "\" + UrduUnicode.alif + ";" +
    UrduRoman.bay + "\" + UrduUnicode.bay + ";" +
    ...
```
public static final Transliterator roman_to_unicode =
        Transliterator.createFromRules("RomanUrdu-Unicode",
        roman_to_Unicode_rules, 0);

These transliterators are used to transliterate the text in following way wherever needed:

String romanText = Transliterator_ur.unicode_to_roman.transliterate("Unicode Text");

and

String unicodeText = Transliterator_ur.roman_to_unicode.transliterate("Roman Text");

A graphic user interface is designed so a user can transliterate Urdu script to Roman and
Roman to Urdu script easily. In both cases a textbox displays the Unicode values for the
written text.

3.5.2 Urdu Extractor

Lexicon is an important part of a morphological system. Unlike English or any ASCII based
language there is less Urdu literature available in digital format. Even the literature found in
digital format is often saved in graphical picture (jpeg, gif) which is not useful for any text
processing. However the wide acceptance and use of Unicode characters opens a new window
of possibilities for text processing for Urdu language. Today lot of Unicode based Urdu
websites can be found on Internet. We even manage to find two book banks for Urdu literature
in Unicode format.

An Urdu Extractor program has been developed to save Urdu Unicode based web pages, which
has further been used for the extraction of lexicon automatically by appropriate techniques and
tools (section 5.2).

Urdu Extractor program is written in Java. Java not only supports Unicode but also save string
and char data in the form of Unicode characters. To extract text from web pages, the first thing
to consider was to remove all HTML tags available in the page. To do so, a class
HTMLProcessor is defined and further a method named “public String
extractTextfromHTMLFile(String uriStr)”is implemented. This method takes a web address,
removes all HTML tags and returns the actual text written on that page. This method also
removes the text other then Urdu Unicode text, only permitting Urdu Unicode characters
defined in UrduUnicode class (section 3.5.1). Therefore if a web page contains text written in
ASCII Roman (e.g. English) and in Unicode Urdu, only Urdu text is returned. This is achieved
further by using Java Regular Expressions (java.util.regex package).
Furthermore, following two useful methods are defined in HTMLProcessor class to save the
extracted text in appropriate format.

public void writeHTMLFile(String text, String file)
public void writeTextFile(String text, String file)
A graphic user interface is developed to extract Urdu text in more convenient way.

### 3.5.3 Urdu Keyboard Input Method

As described above, today, most of computer software supports Unicode standards and provide a built-in support for East Asian languages including Arabic, Hebrew and Chinese etc. Such scripts normally have the complex rules for rendering text and are context sensitive with respect to the previous and next letter in the text. Each letter may have more then one glyphs and an appropriate glyph should be displayed in each context. An appropriate font containing all necessary glyphs and a rendering program is required to display such scripts properly. Arabic, Persian and Urdu are examples of such languages.

As mentioned above, all operating systems normally support Unicode Standard these days. Even the graphic controls provided by high level languages provide built-in support for most of the languages (including Arabic). One example of such API is Java Swing package including Java Applets.

Urdu orthographical script is merely considered a sub-script of Arabic by Unicode standards. Which is true to some extent because of the inheritance of most of the letters from Arabic, however it is not true completely, as Urdu contains many new characters which are not found in Arabic or Persian.

Today most of the rendering systems provided by an operating system or by software API, automatically handles the letters found in Arabic but left the other letters of Urdu script untouched. However, this rendering problem can be solved by installing appropriate fonts. There can be found many different fonts for Urdu from different sources in which Center for research and language processing, Pakistan (CRULP, 2001) is the most prominent one. In such case a user must install a keyboard layout suitable for Urdu and an appropriate font. Sometime it is not possible for a user to install them due to many reasons (e.g. root access, lack of technical knowledge, lack of the knowledge of Urdu script etc). To solve this problem an Urdu keyboard input method is developed during this thesis work. By using it, a user can type Urdu without installing Urdu keyboard. An on screen keyboard is also provided to let the user find the appropriate letters. User can type directly from the keyboard or by clicking on the graphic key buttons. A tool tip is provided for every key to display the name of that letter in Roman Urdu. An on-fly transliteration is also provided in a text box. To render and display Urdu correctly in Java Swing controls, an Urdu Font (Nafees Web Naskh) has been embedded inside the application.

### 3.5.4 The Main GUI Application

The above mentioned tools are then combined and further used to build a main GUI application that interface the FM runtime system into Java to provide morphological analysis both in Urdu and Roman. We Interface four kinds of analysis which are provided by FM as a part of its runtime system. A brief description is given below. However for more details, one should read
the User manual of the FM and the Main GUI application from the project home page at (FM, 2004).

**Tagger Mode:**
This mode analyses the given words into their dictionary form and displays their grammatical description.

**C-Trie program:**
It provides similar but faster analysis as compare to Tagger Mode. It is implemented in C language as a part of FM.

**Synthesiser Mode:**
The synthesiser mode takes a word form and generates the complete inflection table of the group from which the word belongs.

**Inflection Mode:**
The inflection mode takes a word form and an interface function defined in CommandsUrdu.hs (such as n1, n2 …for nouns) and either the word exists in the lexicon or not, it displays a complete inflection table for that word form. Therefore this mode could be used to learn the inflections of a certain paradigm.
4 Urdu morphology and its Implementation in Haskell

Urdu is an Amalgamative language (Siddiqi, 1971, page 13). Despite Urdu is an Indo-European languages, its grammar is very complex and is different in many ways from the other Indo-European languages. It is a subject-object-verb language having relatively free word order and rampant pro drop (Butt 2003, page 1). It also shows mixed ergativity therefore in some cases verb agrees with object rather than subject. Urdu also shows morphological causatives in most of the cases. In this chapter we will discuss the Urdu grammar in a morphological perspective. Parts of speech and their paradigms will be explained with sufficient details and then we will discuss our solution that explains them in FM.

4.1 Nouns

Urdu is a weak inflected language. The function of noun in a sentence is usually shown by postpositions (clitics). In Urdu, noun can be inflected in number and case. A noun can be singular or plural. For example:

<table>
<thead>
<tr>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ləɽkɑ, ڃړکا, lrkɑ, boy)</td>
<td>(ləɽke, ڃړکی, LRkE, boys)</td>
</tr>
<tr>
<td>(kɪt̪ɑb, ہکتاب, k(i)tab, book)</td>
<td>(kɪt̪ɑbɛɾn, کتابین, k(i)tabyN, books)</td>
</tr>
</tbody>
</table>

Table 4.1

We have defined a data type Number in TypesUrdu.hs in a following way:

data Number = Singular | Plural

About the case of noun, there exist two opinions:

According to one opinion:

Urdu has three cases for nouns (Schmidt, 1999, page 7); the nominative, oblique and vocative. The nominative case is used for such nouns not followed by any postpositions, typically for the subject case. The oblique case is used for any noun that is followed by a postposition and some nouns have a separate vocative case (mostly appears in imperative sentences).

According to the other school of thought:

Urdu has seven cases which are morphologically realized by seven markers (Butt & King, 2004, Page 4) and according to (Siddiqi, 1971, Page 321) Urdu has eight cases.
In this implementation we respect both opinions as they do not conflict with each other. Therefore, in the Single-word-analysis, a noun inflects only in three cases; nominative, oblique and vocative; while in Combination-analysis, a noun inflects in nine cases.

However for defining the type of “Case” in our Type System, we feel appropriate to take the seven cases mentioned by Butt & King and adding two more cases (oblique, vocative) in it at the end mentioned by Schmidt.

Following are the nine cases with their clitic forms (postpositions):

<table>
<thead>
<tr>
<th>Urdu cases</th>
<th>Clitic Form (Urdu &amp; Roman)</th>
<th>Clitic Form Pronunciation</th>
<th>Morphological effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative (Nom)</td>
<td>Nothing</td>
<td></td>
<td>Nom (no change)</td>
</tr>
<tr>
<td>Oblique (Obl)</td>
<td>Nothing</td>
<td></td>
<td>Nom or its modified form</td>
</tr>
<tr>
<td>Ergative (Erg)</td>
<td>نَے nE</td>
<td>ne</td>
<td>Obl + ne</td>
</tr>
<tr>
<td>Accusative (Acc)</td>
<td>کو kw</td>
<td>ko</td>
<td>Obl + ko</td>
</tr>
<tr>
<td>Dative (Dat)</td>
<td>کو kw, کے kE</td>
<td>ko, ke</td>
<td>Obl + [ko, ke]</td>
</tr>
<tr>
<td>Instrumental (Inst)</td>
<td>سے sE</td>
<td>se</td>
<td>Obl + se</td>
</tr>
<tr>
<td>Genitive (Gen)</td>
<td>کا, کی, کے kE</td>
<td>ka, ki, ke</td>
<td>Obl + [ka, ki, ke]</td>
</tr>
<tr>
<td>Locative (Loc)</td>
<td>مین myN, پر, p(a)r, تک, t(a)k, یا t(a)l(a)k</td>
<td>مین, پر, تک, یا t(a)l(a)k</td>
<td>Obl + [مین, پر, تک, یا t(a)l(a)k]</td>
</tr>
<tr>
<td>Vocative (Voc)</td>
<td>اا(a)E or Nothing</td>
<td>اɛ</td>
<td>اɛ + Obl or modified form of Obl</td>
</tr>
</tbody>
</table>

In the language dependent part of FM, case is defined as a data type in file TypesUrdu.hs, in a following way:

```haskell
data Case = Nominative
            | Oblique
            | Ergative
            | Accusative
```

32
In a similar way, Number is defined as data type in file TypesUrdu.hs, in a following way:

```haskell
data Number = Singular | Plural
```

In this work, the following data types have been defined to represent nouns:

```haskell
data NounForm = NF Number Case
type Noun = NounForm -> Str
```

Parameter types like Number and Case are language dependent parts and to be able to use the common API functions, they should be the valid instances of Param class. We do this in a following way:

```haskell
instance Param Case where values = enum
instance Param Number where values = enum
```

In a similar way NounForm is made a valid instance of Param class.

```haskell
instance Param NounForm where
    values = [NF n c| n<-values, c<-values]
```

### 4.1.1 The overview of Case system:

To understand the use of these cases inside a sentence and to see how these cases can affect a sentence syntactically as well as semantically, here is given a short discussion about them separately:

#### 4.1.1.1 Nominative

The nominative case is fairly straight forward. It can appear with both subject and object. There is no clitic form for it. In the following example ‘اسد’, Asad’ is subjective Nominative and school is Objective Nominative.

1. Asad سکول یی یی ی
   Asad (Noun-Sg-Masc-Nom) School (Noun-Sg-Masc-Nom)
   Asad goes to school

   جاکاکاکاکاکا
   Asad इसका है
4.1.1.2 Ergative

Ergative case can only occur with a subject. (ني, ne) is its clitic form.

2. 

علي نی سیق پڑھا
Ali-ne səbəq peɾhɑ
Ali (Noun-Sg-Masc-Erg) lesson (Noun-Sg-Masc-Nom) read (Past Indef)
Ali read (his) lesson

Lexically, Ergative case seems simple (only an addition of a clitic “ني, ne”), but semantically it can possess many alternation with some other cases. For example Ergative-Nominative alternation which is mentioned by (Butt & King, 2004, page 5):

3. a. 

رام کھانے
ram kʰansa
Ram (Masc-Sg-Nom) cough-(Perf. Masc Sg)
Ram coughed

b. 

رام نی کھانے
ram-ne kʰansa
Ram (Noun-Masc-Erg) cough-(Perf. Masc Sg)
Ram coughed (purposefully)

And the Ergative-Dative alternation in subject which is mentioned by (Butt & king, 2004, page 2):

4. a. 

سیلیم نی سکول جانا پی
saleem-ne school dʒɑnɑ hɛ
Saleem (Sg-Masc-Erg) school (Sg-Masc-Nom) go (Pres.Sg.Masc/Fem) be (Pres 3 Sg)
Saleem wants to go to school

b. 

سیلیم کوسکول جانا پی
saleem-ko school dʒɑnɑ hɛ
Saleem (Sg-Masc-Dat) school (Sg-Masc-Nom) go (Pres.Sg.Masc/Fem) be (Pres 3 Sg)
Saleem has to go to school

The following example shows a relation between Ergative-Accusative cases which has the same semantic meaning:

5. 
4.1.1.3 Accusative

Accusative case appears with object. The clitic form “کو، کو” is used for Accusative. This is the form identical to Dative. (5a) is an example of Accusative case. Here, it would be interesting to talk about a minor difference between dative and accusative case. Distinction between accusative and dative case can be made on the basis of their role in the sentence. Accusative case occurs with the object whereas dative occurs with the second object. Below, in the example (6) (بچہ کو، بچہ کو) is in dative case and above, in (5), (کسی کو، کسی کو) is an accusative case.

One test we can use to distinguish between these two is that accusative case can be optional. Sentence will be correct if we don’t use an accusative marker. For example (5a) can be written as (5b) and it is still a valid sentence. But it is not possible to remove (کو، کو) from (بچہ کو، بچہ کو) in the (6) sentence which is a dative case.

Accusative case can be replaced by an Instrumental case as well (with a minor change in verb) which is not possible in Dative as shown below (Butt & King, 2004, page 8).

6. انجمن نے صدف کو/سیدہ کہانا کہلایا/کہلائیا

انجوم نے صدف کو سیدہ کہانا کہلایا/کہلائیا

انجوم (Fem-Sg-Erg)  صدف (Fem-Sg-Acc/Inst)  کھانا  کھیل-ا-یا/وا-یا

انجوم gave food to eat by sadaf/ Anjum made sadaf to eat food

4.1.1.4 Dative

It has identical form as Accusative. Following is an example explaining the use of dative:

7. اصدائیں علی سے بچہ کو کرکٹ سکھوانے۔

اصدائیں علی سے بچہ کو کرکٹ سکھوانے۔

اصدائیں (Sg-Masc-Erg)  علی (Sg-Masc-Inst)  کرکٹ (Sg-Masc-Dat)  کرکٹ (Noun-Fem-Nom)

sik巧合i
learn (Perfect)
Asad made child to learn cricket from Ali.

8. اسد کو کام کرنا پڑا
asəd-ko kam kər-na pəra
Asad (Sg-Masc-Dat) work (Sg-Masc-Nom) do (Inf-Masc-Sg) fall (Perf-Masc-Sg)
Asad had to work

There is another form of Dative case that can be seen in pronouns. Following are some examples:
9. اسے بھائی!
use bulɑo!
Him (Pron Dat) call (verb)
Call him

10. کیا تمہیں میرے پیغام ملا؟
kya təmhen mera pəɣam mila?
Did you (Pron Dat) my (Pron Gen) message (Masc-Sg-Nom) get (past-Masc)
Did you get my message?

4.1.1.5 Instrumental

Instrumental clitic “ے سے, se” is very important and is used on various places in different sentences. The versatility of this case can be observed in the following examples:

11. میرے بات غورے سے سنو
meri bat ɣor-se suno
My (Pron- Fem) words (Sg-Fem-Nom) carefully (Sg-Inst) listen(verb)
Listen to me carefully

12. ساجدے نے قلم سے لکھا
d٪id-ne qələm-se likʰa
Sad٪id (Sg-Masc-Erg) pen(Sg-Masc-Inst) write (Perf-Masc)
Sad٪id wrote with pen

13. اسد نے لاہور سے فون کیا
asəd-ne Lahore-se fon kia
Asad(Sg-Masc-Erg) Lahore(Masc-Sg-Inst) phone(Sg-Masc-Nom) make(Past-Masc)
Asad called from Lahore.
۴.۱.۱.۶ جنیفہ

جنیفہ کا ایک حالاتی کیفیت ہے جو چیز کی ساکنی یا مالی کی ویمنگ کا اندازہ دیتا ہے۔ اردو میں جنیفہ کا پانچ اقسام کا استعمال کیا جاتا ہے۔ (کہ، کہ) کی ایک صورت ہے جو انعام شکار مذکور والا کا استعمال کرتی ہے۔ (کہ، کہ) کی دوسری صورت جو ایک چیز کے کامان والا کا استعمال کرتی ہے۔ (کہ، کہ) کی تیسری صورت جو ایک چیز کی کامان والا کا استعمال کرتی ہے۔

15. 
یہ یوحن کا کتنہ پیدا
ye yohan-ka kutta he
This (Pron) Johan (Noun-Masc-Gen) dog (Sg-Masc-Nom) be (Pres-Sg)
This is Johan's dog.

16. 
یہ یوحن کی گاو کئی بی
ye Johan-ki gaari he
This (Pron) Johan (Noun-Masc-Gen) car (Sg-Fem-Nom) be (Pres-Sg)
This is Johan's car.

17. 
یہ یوحن کی کپڑہ کئی بی
ye Johan-ke kapre hen
This (Pron) Johan (Sg-Masc-Gen) clothes (Pl-Masc-Nom) be (Pres-Pl)
These are Johan's clothes.

۴.۱.۱.۷ لوکیافت

لوقیافت کا ایک حالاتی کیفیت ہے جو شرکت کا طور پر قسم تھی۔ اردو میں لوقیافت کی پانچ اقسام کا استعمال کیا جاتا ہے۔

18. 
میرے کھیمے کے دور میں چار کمروں کئی
mere qier-men tfar kemre hen
My (Pron) house (Noun-Masc-Loc) four (Noun-Masc) room (Pl-Masc-Nom) be (Pres-Pl)
There are four rooms in my house.

19. 
میز پر کتاب کھہر دی
muz-per katab rek do
Table (Sg-Masc-Loc) book (Sg-Masc-Nom) place (Verb) do (Pres)
Put the book on the table

20. میرے گھر تک چلدیں
merē g̱ər-tək ḥəlo
My (Pron) house (Sg-Masc-Loc) go (verb)
Let’s go to my house

21. میرے گھر تک چلدیں
merē g̱ər -təlek ḥəlo
My (Pron) house (Sg-Masc-Loc) go (verb)
Let’s go to my house

22. اس دیکھت تلے بیہا بی
asƏd ḏəɾ-təle bəṭʰa ɛ
Asad (Sg-Masc-Nom) tree (Sg-Masc-Loc) sit (Pres-Sg-Masc) be (Pres-Sg)
Asad is sitting under the tree

4.1.1.8 Vocative

The vocative case is the case used for identifying the person being addressed. A vocative expression is an expression of direct address, wherein the identity of the party being spoken to is set forth expressly within a sentence. There are two forms for Vocative case in Urdu. Following are some examples for explanation:

23. لڑکے میرے بات سنوں
ləɾko! meri bɑt̪ suno
Boys (Pl-Masc-Voc) my (Pron) words (Noun-Fem-Nom) listen (verb)
Boys! Listen to me

24. اے لڑکا میرے بات سنوں
aɛ-lerkə! meri bɑt̪ suno
O boys (Pl-Masc-Voc) my (Pron) words (Noun-Fem-Nom) listen (verb)
O Boys! Listen to me

25. لڑکے میرے بات سنوں
ləɾkə! meri bɑt̪ suno
Boy (Sg-Masc-Voc) my (Pron) words (Noun-Fem-Nom) listen (verb)
Boy! Listen to me
4.1.1.9 Oblique

An oblique case can appear in any case relationship except the Nominative case or Vocative case. According to (Butt & King, 2004, page 13): “The Oblique is a prerequisite for the ergative, dative, accusative, instrumental, genitive, and locative marking, as well as postpositions, adjectives”

How to make Oblique form:

For a transliterated text, if a singular noun ends with “h” then we replace this “h” with “E” otherwise oblique form becomes similar to the plural nominative form. The plural oblique also gets some changes from its plural nominative form sometimes. They are discussed in this chapter later with the description of each noun group. These changes have been implemented in the RulesUrdu.hs.

For example:

26. خان کارخانے گیا ہے
Khan (Sg-Masc-Nom) factory (Sg-Masc-Obl) go (Perf-Masc-Sg) be (Pres-3-Sg)
Khan has gone to the factory.

The original word for factory is (کارخانہ, کارخانہ, کارکانہ) and its oblique form is (کارخانے, کارخانے, کارکانے)

27. خان لاہور گیا ہے
Khan (Sg-Masc-Nom) Lahore (Sg-Masc-Nom) go (Perf-Masc-Sg) be (Pres-3-Sg)
Khan has gone to Lahore

Following is an example to show how oblique cases behave like a prerequisite for the cases other then Nominative and Vocative case.

28. خان کارخانے تک پہنچ گیا ہے
Khan (Sg-Masc-Nom) factory (Sg-Masc-Loc) walk (verb) go (Perf-Masc-Sg) be (Pres-3-Sg)
Khan has gone to the factory by walk

In the word “کارخانے تک”, “e” shows oblique behavior for a locative case “تک”.

Similarly

29. خان کارخانے تک پہنچ گیا ہے
Khan (Sg-Masc-Nom) factory (Sg-Masc-Loc) walk (verb) go (Perf-Masc-Sg) be (Pres-3-Sg)
Khan has gone to the factory by walk
Here (کارخانے تلک, karxan-e-telak), “e” shows oblique behavior for a locative case “telak”.

4.1.2 Implementation of Noun

Now we will provide the implementation details of noun class for this work:
The first and foremost a very general function that we present is noun_. It is defined in RulesUrdu.hs and following is its definition:

```haskell
noun_ :: DictForm -> DictForm -> DictForm -> DictForm -> DictForm -> Noun
noun_ sg sg_Obl pl pl_Obl sg_Voc pl_Voc (CommonNoun (NF n c)) =
  mkStrWords $ mkNoun sg sg_Obl pl pl_Obl sg_Voc pl_Voc n c
noun_ sg sg_Obl pl pl_Obl sg_Voc pl_Voc _ = mkStr $ rmStr $ unStr $ nonExist

noun_ is using a function named mkNoun. It definition is as follow:

mkNoun:: String -> String -> String -> String -> String -> String -> Number
mkNoun sg sg_Obl pl pl_Obl sg_Voc pl_Voc n c =
  case n of
    Singular -> case c of
      Nominative    -> sg
      Oblique       -> sg_Obl
      Ergative      -> mkFinalForm sg_Obl nE
      Accusative    -> mkFinalForm sg_Obl kw
      Dative        -> mkFinalForm sg_Obl kw
      Instrumental  -> mkFinalForm sg_Obl sE
      Genitive      -> (mkFinalForm sg_Obl ka) ++ " " ++
                       (mkFinalForm sg_Obl ky) ++ " " ++
                       (mkFinalForm sg_Obl kE)
      Locative      -> (mkFinalForm sg_Obl myN) ++ " " ++
                       (mkFinalForm sg_Obl pr) ++" "++
                       (mkFinalForm sg_Obl tak) ++" "++
                       (mkFinalForm sg_Obl t1E) ++ " " ++
                       (mkFinalForm sg_Obl t1k)
      Vocative      -> sg_Voc ++ " " ++
                       (mkFinalFormV2 aE sg_Voc)
    Plural        -> case c of
      Nominative    -> pl
      Oblique       -> pl_Obl
      Ergative      -> mkFinalForm pl_Obl nE
      Accusative    -> mkFinalForm pl_Obl kw
      Dative        -> mkFinalForm pl_Obl kw
      Instrumental  -> mkFinalForm pl_Obl sE
      Genitive      -> (mkFinalForm pl_Obl ka) ++ " " ++
                       (mkFinalForm pl_Obl ky) ++ " " ++
                       (mkFinalForm pl_Obl kE)
      Locative      -> (mkFinalForm pl_Obl myN) ++ " " ++
                       (mkFinalForm pl_Obl pr) ++" "++
                       (mkFinalForm pl_Obl tak) ++" "++
                       (mkFinalForm pl_Obl t1E) ++ " " ++
                       (mkFinalForm pl_Obl t1k)
```

40
posJazm is an important function which is used above. It removes `dʒəzm` where it is necessary.

`dʒəzm` (‘’‘) is a special diacritic symbol that marks the absence of vowel after base consonant (section 3.3.1.2). `dʒəzm` shows following properties in Urdu text:

- The last letter of a word cannot be `dʒəzm`.
- `dʒəzm` cannot be followed by (reamble, 'a), (va'o, 'v), (bəɾi ye, 'y), (ʧʰoɾi ye, 'ι). y).

`posJasm` is applied on all paradigms defined in RulesUrdu.hs.
It is defined in a following way.

```haskell
posJasm :: String -> String
posJasm xxs = unwords $ map pJazm (words xxs)
```

```
pJazm :: String -> String
pJazm xxs = xs5
where
  xs = if ((last xxs)=='\n') then (tk 1 xxs) else xxs
  xs1 = repAllOccr "'a" "a" xs
  xs2 = repAllOccr "'E" "E" xs1
  xs3 = repAllOccr "'y" "y" xs2
  xs4 = repAllOccr "'wN" "wN" xs3
  xs5 = if ((dp 2 xs4) == "'w") then (tk 2 xs4 ++ "'w") else xs4
```
mapAtEnd (x:xs) str  = [x++str]++ mapAtEnd xs str

`mkFinalForm` takes two string variables while first string variable may have more than one string values separated by space characters and the second string value should contain only one string value. This function takes the value of second string variable and concatenates it with each string value separated by space character in the first variable.

For example a noun (`əhsan`, رخصان, a(i)H'san, favor) has two oblique forms (`əhsanət`, احساسات and (`əhsanən`, احساسن). If we want to apply the ergative case (`ne`) on both forms, we could do this by using this function in a following way:

`mkFinalForm "əhsanət əhsanən" "ne"

It will result in the "əhsanət-ne əhsanən-ne" string.

Similarly `mkFinalFormV2` also takes two strings. The variable of the first string contains only one string value and the variable of the second string may have more then one string values separated by space characters. This function takes the value of the first string variable and concatenates it with each string value separated by space character in the second variable.

For example the vocative case of (`əhsan`, رخصان, a(i)H' san, favor) could be represented as follows.

`mkFinalFormV2 "ae" "əhsanət əhsanən"

It will result in the "ae-əhsanət ae-əhsanən" string.

There are two genders in Urdu - masculine and feminine. Urdu nouns may be additionally divided into two groups; marked nouns and unmarked nouns. Marked nouns normally have a gender suffix, while unmarked nouns do not have any morphological information to recognize there gender and must be learnt by heart by non-Urdu speakers.

Nouns can be divided into different classes based on their inflection. We divide nouns on the basis of the ending letters in their singular forms. We started our work by making suitable divisions of nouns into groups which is mentioned as groups by (Siddiqi, page 287, 289, 302-304) and (Schmidt, 1999, page 4). However suitable changes have been done to group words with respect to pure morphological view. This resulted into the following groups.

1. Singular masculine nouns ending with (`əlf, l, a), (ʧọt hə, o, h) and (`ain, ع, e)
2. Singular masculine nouns ending with (`an, ان, aN)
3. Singular masculine nouns not ending with (`əlf, l, a), (ʧọt hə, o, h), (`ain, ع, e) and (an, ان, aN)
4. Singular feminine nouns ending with (ʧọt ye, ى, y)
5. Singular feminine nouns ending with (alɪf, ᵐa), (an, aN), (oŋ, wN)
6. Singular feminine nouns ending with (ya, ŋa, ya)
7. Singular loan feminine nouns ending with (ya, ŋa, ya), exception from the above rule
8. Singular feminine nouns ending with (va'o, ŋ, w)
9. Singular feminine nouns not ending with (alɪf, a, a), (nun ɣʊnnɑ, N), (va'o, ŋ, w), (oŋ, wN)

Following are groups of loan Arabic words:

10. Singular masculine nouns ending with (nun, n), (ar, r, ar)
11. Singular masculine nouns starting with (alɪf, a), (alɪf məddɑ, A) and ending with (re, r, r)
12. Singular feminine nouns ending with (te, t, t)

Following are the groups of loan Persian words:

13. Singular masculine nouns ending with (va'o, ŋ, w)
14. Singular masculine nouns ending with (va'o ʰɤmzɑ, w') or the nouns with no inflection
15. Groups for names (masculine names, feminine names, names of places)

**Implementation of these groups into FM:**

1. Singular masculine nouns ending with (alɪf, a), (ʧɔti he, ŋ, h) and (‘æn, ŋ, e):

This group also includes the Arabic loan nouns ending with (ʧɔti he, ŋ, h).

If a word ends with letter (alɪf, a) or (ʧɔti he, ŋ, h) then:

- To make plural nominative and singular oblique, the last letter is replaced by letter (bəɽi ye, ŧ, E)
- To make plural oblique, the last letter is replaced by string (oŋ, wN) and
- To make plural vocative, the last letter is replaced by letter (va'o, ŋ, w)

If a word ends with (‘æn, ŋ, e) then the rules will remain same as above except that the above mentioned letters will be just added at the end of words without replacing any existing letter.

Following is a table displaying basic forms of this group e.g. (ləƙɑ, l(a)R' ka, boy) and
(burka, بُرَکَة, b(o)r'q(a)e, cloak). The remaining forms are generated by adding appropriate postpositions.

<table>
<thead>
<tr>
<th>Case</th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>ləɽkɑ, l(a)R'ka, b(o)r'q(a)e</td>
<td>ləɽke, l(a)R'ke</td>
</tr>
<tr>
<td></td>
<td>burka, بُرَکَة, b(o)r'q(a)e</td>
<td>burke, بُرَکَة, b(o)r'q(a)e</td>
</tr>
<tr>
<td>Oblique</td>
<td>ləɽke, l(a)R'kE, b(o)r'q(a)e</td>
<td>ləɽkon, l(a)R'kwN</td>
</tr>
<tr>
<td></td>
<td>burke, بُرَکَة, b(o)r'q(a)e</td>
<td>burkon, بُرَکَة, b(o)r'q(a)ew</td>
</tr>
<tr>
<td>Vocative</td>
<td>ləɽke, l(a)R'kE, b(o)r'q(a)e</td>
<td>ləɽko, l(a)R'kw</td>
</tr>
<tr>
<td></td>
<td>burke, بُرَکَة, b(o)r'q(a)e</td>
<td>burko, بُرَکَة, b(o)r'q(a)ew</td>
</tr>
</tbody>
</table>

This group is defined in *RulesUrdu.hs* in the following way:

```haskell
noun_lRka :: DictForm -> Noun
noun_lRka lRka nf =
    noun_sg sg_obl pl pl_obl sg_obl pl_voc nf
    where
      sg     = lRka
      sg_obl = lRk ++ "E"
      pl     = lRk ++ "E"
      pl_obl = lRk ++ "wN"
      pl_voc = lRk ++ "w"
      lRk = if (end == "e") then lRka else (tk 1 lRka)
      end   = dp 1 lRka
```

This function generates the appropriate forms for different cases and then passes them to a more generic function (*noun_*) as parameters.

Then an interface function for this group is defined in *BuildUrdu.hs* in the following way:

```haskell
n1 :: DictForm -> Entry
n1 df = masculine (noun_lRka df)
```

Where *DictForm* is string type and *masculine* is a function which is also defined in *BuildUrdu.hs*. The *masculine* function is applied on such functions that are written for the inflection of masculine words.

```haskell
masculine :: Noun -> Entry
masculine n = noun n Masculine
```

Then this interface function *n1* is added in *CommandsUrdu.hs* to let it behave like a command in the system and lexicon in the following way:

```haskell
commands =
    [("n1", ["lRka"], appl n1),
     ....
] 44
```
Now this interface function \( n1 \) is ready to be used in urdu.lexicon to add new words for this paradigm in a following way:

\[
\begin{align*}
\text{n1 l(a)R'ka} \\
\text{n1 b(o)r'q(a)e} \\
\text{n1 p(a)r'd(a)h}
\end{align*}
\]

The following words use the same inflection for both singular and plural forms and could be taken as exception from this rule:

1) Some Sanskrit masculine words ending with a (singular: (radʒa, raja, king), Plural: (radʒa, raja, king) as well as (radʒe, raja, rajE, kings)
2) Some masculine names for relatives (əbbɑ, a(a)b"a, father), (ʧəʧɑ, c(a)ca), younger uncle) (t̪ɑyɑ, taya, elder uncle), (d̪ɑd̪ɑ, dada, grand father- father’s father), (pʰupʰa, p(o)|hwp|ha, husband of father’s sister), (nɑnɑ, nana, nana, mother’s father) etc
3) Subjective nouns taken from Persian ((dana, a(n), dana, wise), (bina, b(i)yna, clear-sighted), (afna, AX'na, known), (fnasa, X(a)nasa, known) etc)
4) Some non-Prakrit words ((d̪əryɑ, d(a)r'ya, river), (səhrɑ, S(a)H'ra, desert) etc)

2. **Singular masculine nouns ending with (aN).**

If a word ends with string (aN) then:

- To make plural nominative and singular oblique, the last occurrence of letter (əlɪf, ɣ, a) is replaced by letter (ʧʰoʈi ye, ی, y)
- To make plural oblique and plural vocative, the last occurrence of letter (əlɪf, ɣ, a) is replaced by string (oɳ, وں, wN)

Following is a table for the basic forms of this group (kʊɳwaɳ, k(o)n'waN, well).

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>(kʊɳwaɳ, k(o)n'waN)</td>
<td>(kʊɳweɳ, k(o)n'wyN)</td>
</tr>
<tr>
<td>Oblique</td>
<td>(kʊɳwen, k(o)n'wyN)</td>
<td>(kʊɳwoɳ, k(o)n'wwN)</td>
</tr>
<tr>
<td>Vocative</td>
<td>(kʊɳwen, k(o)n'wyN)</td>
<td>(kʊɳwoɳ, k(o)n'wwN)</td>
</tr>
</tbody>
</table>

This group is defined in *RulesUrdu.hs* in following way:
noun_knwaN :: DictForm -> Noun
noun_knwaN knwaN nf =
    noun_ sg sg_obl pl pl_obl sg_obl pl_voc nf
where
    sg      = knwaN
    pl      = knwyN
    sg_obl  = pl
    pl_obl  = knwwN
    pl_voc  = knwwN
    knwyN   = repLstOccr "a" "y" knwaN
    knwwN   = repLstOccr "a" "wN" knwaN

Where repLstOccr is a function defined to replace the last occurrence of a string with another string. Following is its definition:

repLstOccr :: String -> String -> String -> String
repLstOccr v1 v2 xxs = reverse $ repFstOccr (reverse v1) (reverse v2) (reverse xxs)

Following is the definition of repFstOccr function used above.

repFstOccr :: String -> String -> String -> String
repFstOccr v1 v2 "" = ""
repFstOccr "" v2 xxs = xxs
repFstOccr v1 v2 xxs@(x:xs) = s
where
    s = if (begin v1 xxs) then (v2 ++ (snd (splitAt p xxs)))
       else x: (repFstOccr v1 v2 xs)
    p = length v1

The interface function n2 represents this group in the lexicon.

3. Singular masculine nouns not ending with (اَلْیف, یا), (تُحُوریتہ, ہ، ھ), (‘اِئن، ع، ے) and (آن، اَن، اَن) :

For this group of words singular nominative, singular oblique, singular vocative and plural nominative forms remains same and unchanged however letter (و، ں، w) is added to make plural vocative form and string (وں، ں، wN) is added to make plural oblique form.

Following is a table for the basic forms of this group (مَرْض، مردد، m(a)r’d, man).

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>(مَرْض، مردد، m(a)r’d)</td>
<td>(مَرْض، مردد، m(a)r’d)</td>
</tr>
<tr>
<td>Oblique</td>
<td>(مَرْض، مردد، m(a)r’d)</td>
<td>(مَرْضوں، مرددون، m(a)r’dwN)</td>
</tr>
<tr>
<td>Vocative</td>
<td>(مَرْض، مردد، m(a)r’d)</td>
<td>(مَرْضو، مرددو، m(a)r’dw)</td>
</tr>
</tbody>
</table>

This group is defined in RulesUrdu.hs in following way:

noun_mrd :: DictForm -> Noun
noun_mrd mrd nf =
  noun_ sg sg pl pl_Obl sg pl_Voc nf
  where
    sg       = mrd
    pl       = mrd
    pl_Obl   = mrdwN
    pl_Voc   = mrdw
    mrdwN    = mrd ++ "wN"
    mrdw     = tk 1 mrdwN

The interface function n3 represents this group in the lexicon.

For the groups 4 to 9 below, singular nominative, singular oblique and singular vocative forms remain same and unchanged while the following changes occur in rest of the forms:

4. Singular feminine nouns ending with (ʧʰoʈi, ی, y):

- String (an, aN) is added to make plural nominative form.
- Letter (vɑ'o, w) is added to make plural vocative form.
- String (oɳ wN) is added to make plural oblique form.

Following is a table for the basic forms of this group (ləɽki, ِ, l(a)R'ky, girl).

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>(ləɽki, ِ, l(a)R'ky)</td>
<td>(ləɽkiɑɳ, ِ, l(a)R'kyaN)</td>
</tr>
<tr>
<td>Oblique</td>
<td>(ləɽki, ِ, l(a)R'ky)</td>
<td>(ləɽkiɑɳ, ِ, l(a)R'kywN)</td>
</tr>
<tr>
<td>Vocative</td>
<td>(ləɽki, ِ, l(a)R'ky)</td>
<td>(ləɽkiɔ, ِ, l(a)R'kyw)</td>
</tr>
</tbody>
</table>

This group is defined in *RulesUrdu.hs* in following way:

```
noun_krsy :: DictForm -> Noun
noun_krsy krsy nf =
  noun_ sg sg pl pl_Obl sg pl_Voc nf
  where
    sg       = krsy
    pl       = krsyaN
    pl_Obl   = krsywN
    pl_Voc   = krsyw
    krsyaN   = krsy ++ "aN"
    krsywN   = krsy ++ "wN"
    krsyw     = tk 1 krsywN
```

The interface function n4 represents this group in the lexicon.
5. Singular feminine nouns ending with (əlɪf, l, a), (ən, aN), (ən, wN): 

If a word ends with (əlɪf, l, a), then the remaining rules will be following:

- String (ʔeɳ, L—1, y^yN) is added to make plural nominative form.
- Letter (vɑ'o hamza, ؤ, w^) is added to make plural vocative form.
- String (ʔoɳ, ںؤ, w^N) is added to make plural oblique form.

If a word ends with (ən, aN) or (ən, wN), then the remaining rules will be following:

- Last letter of word is replaced by string (ʔeɳ, L—1, y^yN) to make plural nominative form.
- To make a plural vocative form, if word ends with (ən, aN) then the last letter is replaced by letter (vɑ'o hamza, ؤ, w^) otherwise the last letter is replaced by (ʔoɳ, ںؤ, w^N).
- Last letter of a word is replaced by string (ʔoɳ, ںؤ, w^N) to make plural oblique form.

Following is a table of the basic forms of this group e.g. (bla, b(a)la, ghost), (man, maN, mother) or (dʒʊɳ, j(o)wN, a louse).

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nominative</strong></td>
<td>(bla, ُبَّلَأ, b(a)la)</td>
<td>(blaʔeɳ, ُبَّلَأ, b(a)lay^yN)</td>
</tr>
<tr>
<td></td>
<td>(man, ُمَان, maN)</td>
<td>(maʔeɳ, ُمَان, may^yN)</td>
</tr>
<tr>
<td></td>
<td>(dʒʊɳ, ُدِжُوُن, j(o)wN)</td>
<td>(dʒʊʔeɳ, ُدِجُوُن, j(o)wy^yN)</td>
</tr>
<tr>
<td><strong>Oblique</strong></td>
<td>(bla, ُبَّلَأ, b(a)la)</td>
<td>(blaʔoɳ, ُبَّلَأ, b(a)law^N)</td>
</tr>
<tr>
<td></td>
<td>(man, ُمَان, maN)</td>
<td>(maʔoɳ, ُمَان, maw^N)</td>
</tr>
<tr>
<td></td>
<td>(dʒʊɳ, ُدِجُوُن, j(o)wN)</td>
<td>(dʒʊʔoɳ, ُدِجُوُن, j(o)ww^N)</td>
</tr>
<tr>
<td><strong>Vocative</strong></td>
<td>(bla, ُبَّلَأ, b(a)la)</td>
<td>(blaʔo, ُبَّلَأ, b(a)law^)</td>
</tr>
<tr>
<td></td>
<td>(man, ُمَان, maN)</td>
<td>(maʔo, ُمَان, maw^)</td>
</tr>
<tr>
<td></td>
<td>(dʒʊɳ, ُدِجُوُن, j(o)wN)</td>
<td>(dʒʊʔoɳ, ُدِجُوُن, j(o)ww^N)</td>
</tr>
</tbody>
</table>

This group is defined in RulesUrdu.hs in following way:

```
noun_bmj :: DictForm -> Noun
noun_bmj bmj nf = case (dp 1 bmj) of
 "a" -> noun_bla bmj nf
```
While `noun_aN_wN` and `noun_bla` have been defined as follow:

**Singular feminine nouns ending with (əlf, ə, a):**

```haskell
noun_bla :: DictForm -> Noun
noun_bla bla nf =
  noun_ sg sg pl pl_Obl sg pl_Voc nf
  where
    sg  = bla
    pl  = blaYyN
    pl_Obl = blaWN
    pl_Voc = blaW
    blaW   = tk 1 blaWN
    blaYYN = bla ++ "y^yN"
    blaWN  = bla ++ "w^N"
```

**Singular feminine nouns ending with (aɳ, اں, aN) or (oɳ, وں, wN):**

```haskell
noun_aN_wN :: DictForm -> Noun
noun_aN_wN maN nf =
  noun_ sg sg pl pl_Obl sg pl_Voc nf
  where
    ma    = tk 1 maN
    maWN  = ma ++ "w^N"
    sg   = maN
    pl   = ma ++ "y^yN"
    pl_Obl = maWN
    pl_Voc = if ((dp 2 maN)=="aN") then maW else maWN
    maW   = tk 1 maWN
```

The interface function n5 represents this group in the lexicon.

**6. Singular feminine nouns ending with (yɑ, 抢抓, ya):**

There can be found two kind of different inflections from the words that ends with (yɑ, 抢抓, ya).
We encode them by paradigm functions `noun_gRya` and `noun_rya`.

**noun_gRya** has following inflection rules:

- Letter (ɳ, ں, N) is added to make plural nominative form.
- Letter (vɑ’o, ݁, w) is added to make plural vocative form.
- String (oɳ, ںو, wN) is added to make plural oblique form.

It is defined in `RulesUrdu.hs` in following way:

```haskell
noun_gRya :: DictForm -> Noun
noun_gRya gRya nf =
```
noun_rya has following inflection rules:

- String (ʔeɳ, لس, y^yN) is added to make plural nominative form.
- String (ʔoɳ, ؤں, w^N) is added to make plural oblique form and plural vocative form.

It is defined in RulesUrdu.hs in following way:

```
noun_rya :: DictForm -> Noun
noun_rya rya nf =
    noun_ sg sg pl pl_Obl sg pl_Voc nf
    where
        sg = rya
        pl = rya ++ "y^yN"
        pl_Obl = rya ++ "w^N"
        pl_Voc = rya
```

(ریا، ے، r(i)ya, to show off) and (ھیا، ہیا، H(a)ya, modesty) are some examples of this
group.
The interface function n7 represents this group in the lexicon.

7. Singular feminine nouns ending with (wɑ'o, ژ, w):

- String (ʔeɳ, لس, y^yN) is added to make plural nominative form.
- Letter (wɑ'o hamza, ژ, w^) is added to make plural vocative form.
- String (ʔoɳ, ؤں, w^N) is added to make plural oblique form.

This group is defined in RulesUrdu.hs in following way:

```
noun_khshbw :: DictForm -> Noun
noun_khshbw khshbw nf =
    noun_ sg sg pl pl_Obl sg pl_Voc nf
    where
        sg = khshbw
```

(گریا، گریا، g(o)R'ya, doll) and (دیبا، دیبا، D(i)b'ya, small box) are some examples of this
group.
The interface function n6 represents this group in the lexicon.
8. Singular feminine nouns not ending with (ئِ، یا، (نُن گھننا، ن، N)، (وُو، w)، (وُن، wN)):

- String (ئِ، ی، yN) is added to make plural nominative form.
- String (وُو، w، wN) is added to make plural oblique form.
- Letter (وُو، w، w) is added to make plural vocative form.

This group is defined in RulesUrdu.hs in following way:

```hs
noun_ktab :: DictForm -> Noun
noun_ktab ktab nf =
  noun_ sg sg pl pl_Obl sg pl_Voc nf
  where
  pl_Voc  = tk 1 pl_Obl
  sg      = ktab
  pl      = ktab ++ "yN"
  pl_Obl  = ktab ++ "wN"
```

(کتاب، كتاب، book), (گاجر، Gajah, carrot) are some examples of this group.

The interface function n9 represents this group in the lexicon.

Loan Arabic words:

There are many loan Arabic words found in Urdu; sometimes without modifications and sometimes with modifications from their original forms and inflection rules. Most of the loan Arabic words have irregular patterns and inflect poorly according to the rules. However there are few patterns that can be found for loan Arabic words. Following are groups of loan Arabic words that we implemented in this morphology:

9. Singular masculine nouns ending with (نُن، ن، ن)، (عُر، ع، ع)، (عُن، ع، ع)

Some examples of this group are (اِحسان، a(i)H'san, favor), (اِخبار، a(a)K'bar, newspaper), (اِتِبار، a(i)X't(i)har, advertisement) etc
• String (٠ت, ات, at) is added to make plural nominative form.
• Letter (وvO', و, w) is added to make plural vocative form.
• Two plural oblique forms exist for this group. String (ون, ون, wN) and (٠ت, ات, at) is added to make plural oblique form.

and is defined in RulesUrdu.hs in following way:

```haskell
noun_aHsan :: DictForm -> Noun
noun_aHsan ahsan nf =
  noun_ sg sg pl pl_obl sg_voc pl_voc nf
  where
    sg   = ahsan
    pl   = ahsan ++ "at"
    pl_obl = pl ++ " " ++ ahsan ++ "wN"
    sg_voc = ahsan
    pl_voc = ahsan ++ "w"
```

The interface function n10 represents this group in the lexicon.

10. Singular masculine nouns starting with (əlf, ۱, a), (əlf məddɑ, ۱, A) and ending with (re, ۲, r):

Some examples of this group are (اکر, اکر, AK(a)r, at last), (اَمر, اَمر, a(a)mr, order/live forever) etc

• The first most letter (əlf, ۱, a) or (əlf məddɑ, ۱, A) is replaced by string (əwa, ۱, a(ə)a(ə)wa) and last occurrence (if any) of vowel diacritic (Zəbər , ۱, (a)) is replaced by vowel diacritic (Zer , ۱, (i)) to make plural nominative and plural oblique forms.
• Singular vocative and plural vocative forms do not exist.
• Singular nominative and oblique remain same and unchanged.

and is defined in RulesUrdu.hs in following way:

```haskell
noun_AKir :: DictForm -> Noun
noun_AKir akhir nf =
  noun_ sg sg pl pl_obl sg_voc pl_voc nf
  where
    sg   = akhir
    khr_ = if (a==")" then (drop 3 khr) else khr
    r   = dp 1 khr
    kh  = tk 1 khr
    h   = dp 1 kh
    khr_ = if (h==")") then khr else (kh ++ "(i)" ++ r)
```

52
11. Singular feminine nouns ending with (١٠، ت، ١)

This group of nouns normally shows state and condition. Some examples of this group are 
(ندامهٔت، ندامَت، n(a)dam(a)t, embarrassment), (مَحِبَّت، مَحبَّه، m(a)H(a)b"(a)t, love) etc

- String (ئن، ين، yN) is added as suffix to make plural nominative forms.
- Letter (و، w) is added as suffix to make plural vocative.
- String (ورن، ون، wN) is added as suffix to make plural oblique.
- Singular nominative, oblique and vocative remain same and unchanged.

It is defined in RulesUrdu.hs in following way:

```haskell
noun_ndamt :: DictForm -> Noun
noun_ndamt ndamt nf =
    noun_ sg sg pl pl_obl sg_voc pl_voc nf
where
    sg  = ndamt
    pl  = ndamt ++ "yN"
    sg_voc  = ndamt
    pl_obl  = ndamt ++ "wN"
    pl_voc = ndamt ++ "w"
```

The interface function n12 represents this group in the lexicon.

The loan Persian words:

There are many loan Persian words in Urdu as well; sometimes without modifications and
sometimes with modifications from their original forms and inflection rules. Like Arabic loan
words, most of the loan Persian words have irregular patterns and inflect poorly according to
the rules. However there are few patterns that can be found. Following are the groups of loan
Persian words that we implemented in this morphology:

12. Singular masculine nouns end with (ق١و، ۹، w)

Some examples of this group are (آلو، آلو، Al(o)w, potato), (کالو، خالو، Kal(o)w, husband of
mother’s sister) etc

- Letter (ـو، ۹، w^) is added as suffix to make plural vocative.
- String (وئ، ون، w^N) is added as suffix to make plural oblique.
• Singular nominative, oblique and vocative and plural nominative remain same and unchanged.

It is defined in `RulesUrdu.hs` in following way:

```haskell
noun_Alw :: DictForm -> Noun
noun_Alw alw nf = noun_ sg sg pl pl_obl sg pl_voc nf
  where
    sg = alw
    pl = alw
    pl_obl = alw ++ "w^N"
    pl_voc = alw ++ "w^"
```

The interface function n13 represents this group in the lexicon.

13. Singular masculine nouns end with (va’o həmza, ə, w^) or the nouns with no inflection

In this group, all forms of singular and plural remain same and unchanged. Some examples of this group are (b’əʔo, یبوا, b|haw^, price), (təʔo, تاوا, taw^, angry) etc

It is defined in `RulesUrdu.hs` in following way:

```haskell
noun_bhao :: DictForm -> Noun
noun_bhao bhao nf = noun_ sg sg pl pl nf
  where
    sg = bhao
    pl = bhao
```

The interface function n14 represents this group in the lexicon.

14. Groups for names (masculine names, feminine names, names of places):

The inflection for masculine names, feminine names and names of places is very simple. They do not infect at all.

It is defined in `RulesUrdu.hs` in following way:

```haskell
names :: DictForm -> Noun
names df nf = mkName sg nf
  where
    sg = df
```

Three different interface functions are defined for them in the `BuildUrdu.hs` in a following way:

Masculine name:
4.2 Adjectives

In a morphology point of view there are two kinds of adjectives in Urdu

- The one which only inflects in number, case and gender
- The others which only inflect in degree (positive, comparative or superlative)

In this implementation we treat both above mentioned types and they are defined for in TypesUrdu.hs in following way:

```haskell
type Adjective = AdjForm -> Str and
type AdjectiveDeg = AdjDegForm -> Str
```

While `AdjDegForm` and `AdjForm` are defined in the following way:

```haskell
data AdjForm = AdjForm Number Case Gender
data AdjDegForm = AdjDegForm Degree
```

While `Degree` is defined as follow:

```haskell
data Degree = Positive
              | Comparative
              | Superlative
```

**The Adjectives which inflect in number, case and gender:**

Morphologically if the masculine form of an adjective does not end with letter (əlɪf, ə, a), no inflection occurs. But if the masculine form of an adjective ends with letter (əlɪf, ə, a) then it inflects in the following way:

- The feminine form is made by replacing last letter (əlɪf, ə, a) with (ʧọtj ye, ɔ, y).
For Singular masculine, no change occurs while for all other forms of masculine (singular, plural and cases), the last letter (əlf, ə, a) is replaced by (bəɾi, ə, E) and appropriate case-suffix is added.

It is implemented in RulesUrdu.hs in following way:

```haskell
adjective2 :: DictForm -> Adjective
adjective2 df =
    case (last df) of
      'a' -> adj_ df
      _   -> adj_withNoChange df
```

While adj_ and withNoChange are defined as follows:

```haskell
--inflection for adjectives ending with a
adj_ :: DictForm -> Adjective
adj_ df (AdjForm n c g) =
    mkStrWords $ posJazm $ 
    case g of
      Masculine -> mkNoun sg sg_Obl pl pl sg_Obl pl n c
        where
          sg = df
          sg_Obl = d ++ "E"
          pl = d ++ "E"
          d = tk 1 df
      Feminine -> mkNoun sg sg sg sg sg sg n c
        where
          sg = d ++ "y"
          d = tk 1 df

--no inflection for the rest of them
adj_withNoChange :: DictForm -> Adjective
adj_withNoChange df (AdjForm n c g) =
    mkStrWords $ posJazm $ 
    case (n,c,g) of
      _ -> df
```

The interface function adj3 represents this group in the lexicon.

**The Adjectives which inflect in degree:**

There are two ways of making degree forms of adjectives:

1. Persian’s inflectional way:
   No change in adjective positive form. However comparative form is made by suffixing (tər, ə, tr) and superlative by suffixing (tərin, ər, tryn).
2. Urdu’s own non-inflectional way:
Adjective positive form remains same and unchanged. Comparative form is made by affixing phrases such as (bohət̪, ہے, b(a)h(o)t) or (se, سے, sE) and superlative form is made by using phrases such as (səb-se, سب-سے, sb-sE) + adjective positive form.

As an implementation point of view the words with above mentioned inflections are divided into two types: The words that inflect only according to Persian rules and the words that inflect in both, Urdu and Persian rules.

It is implemented in RulesUrdu.hs in following way:

*adjDegree_* is a general function that separates positive, comparative and superlative forms and is used by other defined functions:

```haskell
adjDegree_ :: DictForm -> DictForm -> DictForm -> AdjectiveDeg
adjDegree_ pos comp sup (AdjDegForm degr) =
  mkStrWords $ posJazm $ case degr of
    Positive   -> pos
    Comparative -> comp
    Superlative -> sup
```

*adjective_form1* function is defined for the words that only inflect according Persian rules:

```haskell
adjective_form1 :: DictForm -> AdjectiveDeg
adjective_form1 pos =
  adjDegree_ pos comp sup
  where
    comp  = pos ++ "t(a)r" ++ " " ++ pos ++ "-t(a)r"
    sup   = pos ++ "t(a)ryn" ++ " " ++ pos ++ "-t(a)ryn"
```

The interface function adj1 represents this group in the lexicon.

*adjective_form2* is defined for the words that inflect both according to Urdu and Persian rules:

```haskell
adjective_form2 :: DictForm -> AdjectiveDeg
adjective_form2 pos =
  adjDegree_ pos comp sup
  where
    comp  = pos ++ "t(a)r" ++ " " ++ pos ++ "-t(a)r" ++ " " ++ bohat ++ pos ++ " " ++ sE ++ pos
    sup   = pos ++ "t(a)ryn" ++ " " ++ pos ++ "-t(a)ryn" ++ " " ++ sab_se ++ pos
```

While *bohat* and *sab_se* string values mentioned above.

The interface function adj2 represents this group in the lexicon.
There can be found many irregular complex patterns and phrases used for adjectives in Urdu. However we limit our discussion and implementation with the adjective forms discussed above.

4.3 Verbs

The Urdu verbs are very complex as compared to the other word classes. Urdu verb inflects in tense, mood, aspect, gender and number. Many verb auxiliaries (helping verb) are also used to represent a correct tense, mood and aspect of a verb. Furthermore verb auxiliaries also infect in tense, mood, aspect, gender and number as a normal verb.

Urdu verb shows causative behavior (direct & indirect). These causatives are normally made from a basic stem form of the verb. Mostly each verb has only one basic stem form. Some of such stem forms could be following:

Intransitive stem form: (bən, بن, be made), (ʊʈ, اُت, rise), (bəʈ, بٹ, sit) etc
Transitive stem form: (kɑʈ, ٹک, cut), (xərid̪, ضرید, buy), (mil, مل, meet) etc
Verbs of motion stem form: (dʒɑ, چو, go), (ɑ, آ, come), (bʰag, بھاگ, run) etc
Verbs of perception stem form: (d̪ek, دیک, see), (ɖər, دار, fear) etc

In other words, in general, for each verb, there exists at least one stem form (Intransitive, transitive etc). This basic stem form then normally forms two other forms (direct & indirect causatives) of that verb. These generated forms (verbs) can have similar or different meanings from each other. These three forms are actually regular verbs and inflect in tense, mood, aspect, gender and number.

For example consider a verb (bən, بن, be made):

Basic Infinitive form: (bənnɑ, بنا, be made)
Direct Causative Infinitive form: (bənɑnɑ, بنانا, to make/cause to make)
Indirect Causative Infinitive form: (bənwɑnɑ, بنوانا, cause to be made)

(bənnɑ, بنانا, be made), (bənɑnɑ, بنانا, to make/cause to make) and (bənwɑnɑ, بنوانا, cause to be made) are three regular verbs and inflect in tense, mood, aspect, gender and number.

4.3.1 Verb categories:

In the perspective of morphology, we divide verbs in the following categories:

1) Verbs: direct causative & indirect causative cannot be made from their basic stem form
2) Verbs: direct causative & indirect causative can be made from their basic stem form
   i) Made by rules
   ii) All Irregulars
3) Verbs: only direct causative can be made from their basic stem form
4) Verbs: only indirect causative can be made from their basic stem form

To capture this abstraction we define verbs in TypesUrdu.hs in the following way:

1) type Verb_Auxilary = Verb_AuxilaryForm -> Str
   (This type is defined for verb auxiliaries)

2) type Verb = VerbForm -> Str
   (This type is defined for such verbs that cannot make direct & indirect causative)

3) type Verb1 = VerbForm1 -> Str
   (This type is defined for such verbs that can make both direct & indirect causative)

4) type Verb2 = VerbForm2 -> Str
   (This type is defined for such verbs that can only make direct causative)

5) type Verb3 = VerbForm3 -> Str
   (This type is defined for such verbs that can only make indirect causative)

In line (1), type `Verb_AuxilaryForm` is defined to capture the inflection of auxiliaries in a following way:

```
data Verb_AuxilaryForm = VA Tense_axiliary Person Number Gender   |
                      VA_Root       |
                      VA_Inf       |
                      VA_Inf_Obl
```

Where `VA` is constructor name and `Tense_axiliary` is defined as follows:

```
data Tense_axiliary= Past   | Present  | Future  |
                   Imperative | Subjunctive | Perfective |
                   Imperfective
```

While Person is defined as below:

```
data Person = FirstPerson
               | SecondPerson_VeryCasual
               | SecondPerson_Familiar
               | SecondPerson_Respect
               | ThirdPerson_Near
               | ThirdPerson_Distant
```
In Urdu, Second Person has three forms; to be very casual (تُو، تُو), to be casual but a little formal (تُم, تُم) and to be respectful (آپ, آپ). While for a third person (یہ, یہ) and (وہ) are used which can be translated as “this” and “that” respectively.

The first line in *Verb AuxiliaryForm* is used for the conjugation of auxiliaries like (ہونا, ہونا). So a verb auxiliary inflects in:

- **Gender** (Masculine, Feminine)
- **Number** (Singular, Plural)
- **Person** (FirstPerson, SecondPerson, ThirdPerson)
- **Aspect, Tense and Mood** (Past, Present, Future, Imperative, Subjunctive, Perfective, Imperfective)

While in the last three lines of *Verb AuxiliaryForm*, constructors *VA Root, VA Inf, VA Inf Obl* represents root, infinitive, and infinitive oblique forms of verb auxiliary respectively.

In line (2), type *VerbForm* is defined in a following way:

```haskell
data VerbForm = VF VAnalysis | Inf | Root | Inf_Obl
```

The last three lines of *VerbForm*, constructors *Root, Inf, Inf_Obl* represents root, infinitive, and infinitive oblique forms of verb respectively.

While *VAnalysis* is defined as follows:

```haskell
data VAnalysis =  SingleWA BasicVerbForm Person Number Gender | CombinationA Tense Person Number Gender
```

In the first line of *VAnalysis* type, *SingleWA* is a constructor name. This form is used to generate Single-word-analysis. While *BasicVerbForm* is another user defined type that shows the mood of a verb, defined as follows:

```haskell
data BasicVerbForm =  Subjunctive | Perfective | Imperfective
```

In a similar way, in the second line of *VAnalysis* type, *CombinationA* is a constructor for providing Combination-analysis. While *Tense* is defined following:

```haskell
data Tense= PastIndefinite | PastImperfective | PastImperfectiveContinuous | PastImperfectiveHabitual | PastImperfectiveHabitualContinuous
```
This is used for the conjugation of regular verbs.

To recap, for the inflection of verbs, we provide two kind of analysis that inflects in a following way:

**For Single-word-analysis a verb inflects in:**

- **Gender** (Masculine, Feminine)
- **Number** (Singular, Plural)
- **Person** (FirstPerson, SecondPerson, ThirdPerson)
- **BasicVerbForm** - synonym of Mood & Tense (Subjunctive, Perfective, Imperfective)

**For Combination-analysis a verb inflects in:**

- **Gender** (Masculine, Feminine)
- **Number** (Singular, Plural)
- **Person** (FirstPerson, SecondPerson, ThirdPerson)
- **Mood, Aspect & Tense:**
  - **Past:** Indefinite, Imperfective, ImperfectiveContinuous, ImperfectiveHabitual, ImperfectiveHabitualContinuous, PerfectiveImmediate, PerfectiveDistant, Presumptive, Conditional
  - **Present:** Indefinite, ImperfectiveContinuous, Imperative
  - **Future:** Indefinite, ImperfectiveContinuous, Presumptive

In line (3), type *VerbForm1* is defined in a following way:

```haskell
data VerbForm1 = VF_VAnalysis
                  Caus1 VAnalysis
                  Caus2 VAnalysis
                  Inf_ | Caus1_Inf
                  Caus2_Inf | Inf_Obl
                  Caus1_Inf_Obl | Caus2_Inf_Obl
                  Root_ | Caus1_Root | Caus2_Root
```

As *VerbForm1* type is defined for such verbs that can make both direct & indirect causatives, the first line *VF_VAnalysis* provides both Single-word-analysis and Combination-analysis for
the basic stem of the verb; while *Caus1 VAnalysis* and *Caus2 VAnalysis* provide Single-word-analysis and Combination-analysis for direct and indirect causative forms respectively. *nf_, Caus1 Inf* and *Caus2 Inf* are infinitive forms of basic verb form and its direct & indirect causatives respectively. Similarly *Inf Obl_*, *Caus1 Inf Obl* and *Caus2 Inf Obl* are infinitive oblique forms; while *Root_, Caus1 Root, Caus2 Root* are root forms of basic, direct and indirect causative verb forms.

It could be seen in the following example table for verb (بَنَّ, be made):

<table>
<thead>
<tr>
<th>Intransitive/Transitive/di-transitive</th>
<th>Direct Causative</th>
<th>Indirect Causative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Infinitive</td>
<td>Oblique</td>
</tr>
<tr>
<td>bən</td>
<td>bənna</td>
<td>bəne</td>
</tr>
<tr>
<td>بن</td>
<td>بُنَا</td>
<td>بَنَّي</td>
</tr>
</tbody>
</table>

Here one thing that we want to mention is that the root forms not only participate in making all other inflection forms but also fulfils the following forms of the verbs:

*bən-kər, bən-ke* (بَن كَر، بن كَي), *bənə-kər, bənə-ke* (بَنَّ كَر، بَنَّ كَي) and *bənwə-kər, bənwə-ke* (بَنَوَّ كَر، بَنَوَّ كَي).

In line (4, 5), type *VerbForm2* and *VerForm3* are defined in a following way:

```haskell
data VerbForm2 = VF2 VAnalysis      | VCAus1 VAnalysis      | Inf2 | VCAus1 Inf | Inf_Obl2 |
               |                     | VCAus1 Inf_Obl | Root2 | VCAus1 Root |

data VerbForm3 = VF3 VAnalysis      | VCAus2 VAnalysis      | Inf3 | VCAus2 Inf | Inf_Obl3 |
               |                     | VCAus2 Inf_Obl | Root3 | VCAus2 Root |
```

These definitions are similar to the definition given for *VerForm1*.

In this section we will not show the complete implementation as the code generating the inflection is very huge due to the big type system that we generated for verbs.

We start our implementation of verbs by defining function according to the categories of verbs defined above:

### 4.3.2 Group 1:

The verbs have only basic stem form (intransitive/transitive etc), while direct causative & indirect causative cannot be made:
regVerb1 :: String -> Verb
regVerb1 infin = mkRegVerb root infin
    where
        root = tk 2 infin

While mkRegVerb is defined as:

In each line (mkStrWords $ posJazm $) has been exempted to make the code more readable.

mkRegVerb :: DictForm -> DictForm -> Verb
mkRegVerb root infin (Root)    = root
mkRegVerb root infin (Inf)    = infin
mkRegVerb root infin (Inf_Obl)   = ((tk 1 infin)++"E")
mkRegVerb root infin (VF (SingleWA bvf p n g))  =
        mkSingleWA root bvf p n g
mkRegVerb root infin (VF (CombinationA tense person number gender))=
        mkCombinationAnalysis root tense person number gender

One thing that we want to mention is that, each line of code represents a particular case at the left hand side and the right hand side defines the solution for that particular case. For example in the code:

mkRegVerb root infin (Inf) = mkStrWords $ infin

The (Inf) is a particular constructor (case) of data type Verb so in the line above it is stated that the “mkStrWords $ infin” should be executed only if this particular form appears (Inf). While mkSingleWA is a general function, responsible for Single-word-analysis and mkCombinationAnalysis is a general function responsible for Combination-analysis.

The interface function v1 is defined for this group.

4.3.3 Group 2:

The direct & indirect causatives can be made from the basic stem form (intransitive/transitive etc), by rules. These rules are taken from (Siddiqi, page 335-6).

All the subgroups in group 2, uses the following general function in which six parameters are passed for correct inflection of the basic form, direct causative and indirect causative forms:

Only a part of its definition is shown below where in each line (mkStrWords $ posJazm $) has been exempted to make the code more readable:

mkGenVerb :: DictForm -> DictForm -> DictForm -> DictForm -> DictForm -> DictForm -> Verb1
mkGenVerb r r1 r2 vf caus1 caus2 (Root_)= r
mkGenVerb r r1 r2 vf caus1 caus2 (Inf_)= vf
mkGenVerb r r1 r2 vf caus1 caus2 (Inf_Obl_)= ((tk 1 vf)++"E")

------
------
4.3.3.1 Group 2.1:

If the root form of a basic stem verb does not end with (əlɪf, ə, a), (wɑ'o, w) and (ʧhə[ɪ] ye, ی, y) then to make causatives:

- (əlɪf, ə, a) and (wɑ, w, wa) is added in between the root form and the suffix of infinitive form.
- If the first letter is followed by (æe, ی, (a)y) then it will be replaced by vowel (i, ی, (i)) in both causative forms.
- If the first letter is followed by (i, ی, y) then it will be deleted in both causative forms.

It is implemented in the following way:

```haskell
regVerb2 :: String -> Verb1
regVerb2 vInf =
    mkGenVerb root r1 r2 vInf caus1 caus2
    where
        root = tk 2 (vInf)
        caus1 = (rem_y root) ++ "ana"
        caus2 = (rem_y root) ++ "wana"
        r1 = tk 2 caus1
        r2 = tk 2 caus2

rem_y :: String -> String
rem_y str = st
    where
        b  = take 1 str
        yth = drop 1 str
        y1 = take 4 yth
        y2 = take 1 yth
        th  = if (y1=="(a)y") then (drop 5 str) else (drop 2 str)
        st  = if (y1=="(a)y") then (b ++ "(i)"++th)
             else if (y2=="y") then (b ++ th)
             else str
```

Interface function v2 is defined for this group.

(پرٰنا، p(a)R|hna, to read) and (لئاکنا، لئاکنا، l(a)T(a)kna, to be put off) are some examples. Following is the inflection:

<table>
<thead>
<tr>
<th>Root form</th>
<th>Intransitive/Transitive infinitive form</th>
<th>Direct Causative infinitive form</th>
<th>Indirect Causative infinitive form</th>
</tr>
</thead>
<tbody>
<tr>
<td>(پرٰ، p(a)R</td>
<td>h)</td>
<td>(پرٰنا، پرٰنا، p(a)R</td>
<td>hna)</td>
</tr>
<tr>
<td>(لئاک، لئاک، l(a)T(a)k)</td>
<td>(لئاکنا، لئاکنا، l(a)T(a)kna)</td>
<td>(لئاکنا، لئاکنا، l(a)T'kana)</td>
<td>(لئاکنا، لئاکنا، l(a)T(a)kwana)</td>
</tr>
</tbody>
</table>
4.3.3.2 Group 2.2:

To make direct causative from intransitive/transitive, (əlf, ə, a) is added before the last letter of the root form and for indirect causative form (wa, wa, wa) is added further.

It is implemented in the following way:

```haskell
regVerb3 :: String -> Verb1
regVerb3 vInf =
    mkGenVerb root r1 r2 vInf caus1 caus2
    where
        root = tk 2 (vInf)
        caus1 = (add_a root) ++ "na"
        caus2 = root ++ "wana"
        r1 = tk 2 caus1
        r2 = tk 2 caus2

add_a :: String -> String
add_a str = st
    where
        al = dp 5 str
        nik = if (((countV str)==0) && ((countDiacret str) == 0)) then tk 1 str
            else if ((countV al==0) && (countDiacret al==1)) then (tk 2 str)
            else if ((countV al==1) && (countDiacret al==0)) then (tk 4 str)
            else if ((countV al==1) && (countDiacret al==1)) then (tk 5 str)
            else (tk 1 (remDiacret str))
        l  = if ((countV str)==0) && (countDiacret str==0)) then dp 1 al
            else if ((countV al==0) && (countDiacret al==1)) then (dp 2 al)
            else if ((countV al==1) && (countDiacret al==0)) then (dp 4 al)
            else if ((countV al==1) && (countDiacret al==1)) then (dp 5 al)
            else (dp 1 (remDiacret al))
        l_ = remV $ remDiacret l
    st = nik ++ "a" ++ l_
```

The occurrence of vowels and diacritics make the definition of “add_a” functions a little complex. In this function we check all the possible forms with respect to the vowels and diacritics and make a final form of direct causative.

Interface function v3 is defined for this group.

\[
\begin{align*}
\text{(nikəlna, n(ik)a(l)'na, to come out), (kəña, kənna, kT'na, to cut)} & \quad \text{and (sən'ḇhəl, sən'ḇhəl'na, to recover oneself)} \quad \text{are some examples. Following is the inflection:}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Root form</th>
<th>Intransitive/Transitive infinitive form</th>
<th>Direct Causative infinitive form</th>
<th>Indirect Causative infinitive form</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nikəl, n(ik)a(l))</td>
<td>(nikəlna, n(ik)a(l)'na)</td>
<td>(nikəl'na, n(ik)a(l)'na)</td>
<td>(nikəlwana, n(ik)a(l)'wana)</td>
</tr>
<tr>
<td>(sən'ḇhəl, sən'ḇhəl'na)</td>
<td>(sən'ḇhəl'na, sən'ḇhəl'na)</td>
<td>(sən'ḇhəl'na, sən'ḇhəl'na)</td>
<td>(sən'ḇhəlwana, sən'ḇhəlwana)</td>
</tr>
</tbody>
</table>
### 4.3.3.3 Group 2.3:

For all other irregular forms, a function `mkVerbCaus12` is defined:

```haskell
mkVerbCaus12 :: String -> String -> String -> Verb1
mkVerbCaus12 vInf caus1_inf caus2_inf =
    mkGenVerb root r1 r2 vInf caus1_inf caus2_inf
    where
        root  = (tk 2 vInf)
        r1    = (tk 2 caus1_inf)
        r2    = (tk 2 caus2_inf)
```

In this function we provide, the basic, direct and indirect causative forms as argument. As in Urdu, the conjugation of verbs is very regular; a complete inflection can be built with these three forms. An interface function `v4` is defined for this group and they are added in lexicon with the following signature:

- `v4 c(o)r'na c(o)rana c(o)r'wana`
- `v4 m(i)l'na m(i)lana m(i)l'wana`

### 4.3.4 Group 3:

Only direct causative can be made from the basic stem form (intransitive/transitive etc). A function `mkVerbCaus1` is defined for such forms:

```haskell
mkVerbCaus1 :: String -> String -> Verb2
mkVerbCaus1 vInf caus1_inf =
    mkGenCaus1 root r1 vInf caus1_inf
    where
        root  = (tk 2 vInf)
        r1    = (tk 2 caus1_inf)
```

An interface function `v5` is defined for this group and they are added in lexicon in the following signature:

- `v5 jagna jgana`
- `v5 shna sharna`
4.3.5 Group 4:

Only indirect causative can be made from the basic stem form (intransitive/transitive etc). A function `mkVerbCaus2` is defined for such forms:

```haskell
mkVerbCaus2 :: String -> String -> Verb3
mkVerbCaus2 vInf caus2_inf =
    mkGenCaus2 root r2 vInf caus2_inf
    where
        root  = (tk 2 vInf)
        r2  = (tk 2 caus2_inf)
```

An interface function `v6` is defined for this group and they are added in lexicon in the following signature:

```haskell
v6 bycna bycwana
v6 jancna jancwana
```

4.3.6 Verb conjugations:

Urdu verbs demonstrate very regular verb conjugation with an exception of following five verbs. (`honɑ`, بوتا, hwna, be), (`kərnɑ`، َْ، k(a)r'na, to do), (`dena`, دينا, dyna, to give), (`lena`, لينا, lyna, to take), (`jɑnɑ`, جانا, jana, to go) (Schmidt, 1999, page 92).

These five verbs are used frequently in Urdu sentences sometimes alone and sometimes as helping verbs (auxiliaries).

First we describe a part of the conjugation (for Past tense) of (`honɑ`, بوتا, hwna, be). Complete table is given in Appendix B.

It is described as an auxiliary form of verb which is treated with Constructor (VA Tense Axiliary Person Number Gender) in type system:

<table>
<thead>
<tr>
<th></th>
<th>Past</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First Person</td>
</tr>
<tr>
<td></td>
<td>Casual</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>تُها</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>تُها تُها</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>تُها تُها</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>تُها تُها تُها</td>
</tr>
</tbody>
</table>

67
The inflection of verb auxiliary (ہونا, بیمار, hwna) is defined in RulesUrdu.hs as this auxiliary plays an important role to decide the tense of a verb. As mentioned above, it does not follow any regular pattern in its inflection; therefore the whole conjugation table for (ہونا, بیمار, hwna) is hard coded into RulesUrdu.hs. A partial implementation is shown below in which an inflection for Past tense is shown:

```haskell
mkAux_hona :: Verb_Auxiliary
mkAux_hona (VA_Root)= mkStr $ root_ho
mkAux_hona (VA_Inf)= mkStr $ infin_hona
mkAux_hona (VA_Inf_Obl)= mkStr $ infin_honE
mkAux_hona (VA tense person number gender) =
  mkStrWords $ case tense of
    Past -> case person of
      SecondPerson_Familiar -> case number of
        Singular -> case gender of
          Masculine -> past_thE
          Feminine -> past_thi ++ " " ++ past_thyN
        Plural -> case gender of
          Masculine -> past_thE
          Feminine -> past_thyN
      SecondPerson_Respect -> case number of
        Singular -> case gender of
          Masculine -> past_thE
          Feminine -> past_thi ++ " " ++ past_thyN
        Plural -> case gender of
          Masculine -> past_thE
          Feminine -> past_thyN
      _ -> case number of
        Singular -> case gender of
          Masculine -> past_tha
          Feminine -> past_thi
        Plural -> case gender of
          Masculine -> past_thE
          Feminine -> past_thyN
```

An instance of this function is given in DictUrdu.hs which is a lexicon for closed classes.

In a similar way the above mentioned five verbs including (ہونا, بیمار, hwna, be) are also hard coded in the RulesUrdu.hs. The reason to add (ہونا, بیمار, hwna, be) again is the fact that it also inflects as a normal verb. The difference between the auxiliary (ہونا, بیمار, hwna, be) and verb (ہونا, بیمار, hwna, be) could be seen with the following example:

وہ بیمار بیمار
wo bimar hoa hoga
He(Pron) sick(verb) be (Verb Perf) be (verb aux Fut)
He might have been sick
4.3.6.1 Single-word-analysis of a verb

Similarly a part of conjugation for verb (لاتانة، lana, to bring) is shown below while complete table is given in Appendix B.

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ləyɑa lɑnɑ</td>
<td>ləyɑa lɑnɑ</td>
<td>ləʔe lɑnɑ</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>ləʔi lənɑ</td>
<td>ləʔi lənɑ</td>
<td>ləʔi lənɑ</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>ləʔe lənɑ</td>
<td>ləʔe lənɑ</td>
<td>ləʔe lənɑ</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>ləʔiən lənɑ</td>
<td>ləʔiən lənɑ</td>
<td>ləʔiən lənɑ</td>
</tr>
</tbody>
</table>

A function *mkSingleWA* is defined for this conjugation that is then further reused by *mkGenVerb*, *mkRegVerb*, *mkGenCaus1* and *mkGenCaus2* functions which are the general functions for all verb groups, generating their inflections according to the conjugations.

A partial implementation is show below in which an inflection for Perfective tense is shown:

```haskell
mkSingleWA :: String -> BasicVerbForm -> Person -> Number -> Gender -> String
mkSingleWA root bvf p n g =
  case bvf of
    Perfective -> mkPastInd root p n g
    _ -> _

Where *mkPastInd* function is defined as follows:

```haskell
mkPastInd :: String -> Person -> Number -> Gender -> String
mkPastInd root person number gender =
  case person of
    FirstPerson  -> case number of
      Singular   -> case gender of
        Masculine -> mkEnding1 b root "yə" "a"
        Feminine  -> mkEnding1 b root "y" "y"
      Plural     -> case gender of
        -> mkEnding1 b root "y^E" "E"
    SecondPerson_VeryCasual -> case number of
      Singular   -> case gender of
        Masculine -> mkEnding1 b root "yə" "a"
        Feminine  -> mkEnding1 b root "y" "y"
      Plural     -> case gender of
        Masculine -> mkEnding1 b root "y^E" "E"
        Feminine  -> mkEnding1 b root "y" "yN"
    SecondPerson_Familiar -> case number of
```

69
case gender of
  Masculine -> mkEnding1 b root "y^E" "E"
  Feminine -> mkEnding1 b root "y^yN" "yN"

SecondPerson_Respect -> case number of
  Singular -> case gender of
    Masculine -> mkEnding1 b root "y^E" "E"
    Feminine -> mkEnding1 b root "y^yN" "yN" ++ "" ++
      mkEnding1 b root "y^y" "y"
  Plural -> case gender of
    Masculine -> mkEnding1 b root "y^E" "E"
    Feminine -> mkEnding1 b root "y^yN" "yN"

  case number of
  Singular -> case gender of
    Masculine -> mkEnding1 b root "ya" "a"
    Feminine -> mkEnding1 b root "y^y" "y"
  Plural -> case gender of
    Masculine -> mkEnding1 b root "y^E" "E"
    Feminine -> mkEnding1 b root "y^yN" "yN"

where
  t = dp 1 root
  b = inStr t ["A","a","w"]

The use of \textit{mkPastInd} function \textit{mkSingleWA} function for Perfective case also shows our concern to reuse of the code wherever possible.

As it can be seen that \textit{mkEnding1} is used exclusively in the function above. This function is generic that can be used for applying custom endings for different forms of a word according to some pattern or conditions.

For example:

\begin{verbatim}
  mkEnding1 b root " y^yN" "yN"
  where
  t = dp 1 root
  b = inStr t ["A","a","w"]
\end{verbatim}

In above statements it is stated that if a word ends with (əlɪf-məddə, ī, A), (əlɪf, ī, a) or (və'o, ī, w) in root form then add (ʔenₐ, y^yN) otherwise (enₐ, yN).

\subsection*{4.3.6.2 Combination-analysis of a verb}

The implementation for the Combination-analysis is similar to the Single-word-analysis. To demonstrate a part of the conjugation (past indefinite) is shown below:

<table>
<thead>
<tr>
<th>Past Indefinite</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>First Person</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Sg.</strong>&lt;br&gt;Masc</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
The part of the definition of function \texttt{mkCombinationAnalysis} is shown below. It is responsible for the composite analysis.

\texttt{mkCombinationAnalysis:: String -> Tense -> Person -> Number -> Gender -> String}

\texttt{mkCombinationAnalysis root tense person number gender =}
\begin{verbatim}
case tense of
  PastIndefinite -> mkPastInd root person number gender
  PastImperfective -> mkPastImperf root person number gender
  PastPerfectiveDistant -> mkPastPerfDist root person number gender
  PastPresumptive -> mkPastPresumpt root person number gender
\end{verbatim}

Similarly a part of the definition of \texttt{mkPastInd} is show below which generates the inflection for Past Indefinite tense.

\texttt{mkPastInd :: String -> Person -> Number -> Gender -> String}

\texttt{mkPastInd root person number gender =}
\begin{verbatim}
case person of
  FirstPerson    -> case number of
     Singular    -> case gender of
        Masculine -> mkEnding1 b root "ya" "a"
        Feminine  -> mkEnding1 b root "y^y" "y"
     Plural      -> case gender of
        Masculine -> mkEnding1 b root "y^E" "E"
        Feminine  -> mkEnding1 b root "y^N" "N"
  SecondPerson_VeryCasual -> case number of
     Singular    -> case gender of
        Masculine -> mkEnding1 b root "ya" "a"
        Feminine  -> mkEnding1 b root "y^y" "y"
     Plural      -> case gender of
        Masculine -> mkEnding1 b root "y^E" "E"
        Feminine  -> mkEnding1 b root "y^yN" "yN"
\end{verbatim}

\begin{verbatim}
where
  t = dp 1 root
  b = inStr t ["A","a","w"]
\end{verbatim}
4.4 Adverbs

For the implementation of Adverbs, we divided them into following categories as mentioned by (Schmidt, 1999, page 51):

- Adverbs of Time
- Adverbs of place
- Adverbs of manner
- Adverbs of degree
- Model Adverbs

First we start from the data type that we define for Adverbs.

```haskell
data AdverbForm = AdverbForm
type Adverb = AdverbForm -> Str
```

Since Adverbs do not inflect therefore we combined all above mentioned categories and represent them with AdvForm constructor.

Following are some example Adverbs:

**Adverbs of Time:** (hmeʃə, همیشه, forever), (kəl, كل, tomorrow/yesterday), (əksər, أكثر, often), (əb, اب, now), (kəb, کب, when) are some examples.

**Adverbs of place:** (bɑhir, بایر, outside), (əndər, اندر, inside), (kərib, قريب, ) and (d̪ur, دور, far) etc

**Adverbs of manner:** (yʊɳ, ون, thus), (kiʊɳ, کن, why) etc

**Adverbs of degree:** (bɽɑ, ای, big) etc

An interface function `adj1` is defined for adverbs.

4.5 The Closed classes

4.5.1 Pronouns

A pronoun inflects in number, person, gender, case and some times not at all. Their inflection is quite irregular however we tried to define general functions that different pronouns can reuse.

4.5.1.1 Personal pronouns:

The Personal Pronouns inflect in number, person and case. Except some relative pronouns, pronouns do not have any gender in Urdu language.
We define the following data type for personal pronouns:

```haskell
type PersPron = PersPronForm -> Str
```

where `PersPronForm` is defined as follows:

```haskell
data PersPronForm = PP Number Person Case
```

The inflection for personal pronouns is given in Appendix C. A function `pronPersonal` is defined in RulesUrdu.hs for them.

### 4.5.1.2 Demonstrative Pronoun:

Demonstrative pronouns stand in for a person, place or thing that must be pointed to. They inflect in number and case.

We define the following data type for them:

```haskell
data DemPronForm = DP Number Case
type DemPron = DemPronForm -> Str
```

(ये, یہ, this) and (وہ, that) are demonstrative pronouns in Urdu. A function `demonsPron` is defined for them.

### 4.5.1.3 Reflexive Pronoun:

(خود, آپ, اپنے آپ, آپنے مین, اپنے آپے, اپنے آپے, اپنے آپے, اپنے آپے) are reflexive pronouns. They do not inflect at all.

A very simple data type is defined in a following way:

```haskell
data RefPronForm = RefP
type RefPron = RefPronForm -> Str
```

This type could also be defined easily as “`type RefPron = String -> Str`”. The reason to go for above mentioned declaration is to be able to associate a custom string value with these words at the time of analysis. It is done by adding the following line of code into BuildUrdu.hs:

```haskell
instance Dict RefPronForm where category _ = "Reflexive Pronoun"
```

A function `pronReflex` is defined for them.
4.5.1.4 Interrogative pronoun:

Interrogative pronouns basically stand in for the answer to the question being asked. When they are not acting as interrogative pronouns, some may act as relative pronouns. Some of them inflect in number and case. While some of them do not inflect at all.

The one who inflects, their inflection is completely similar to the inflection of relative pronoun given in Appendix C if we replace (dʒo, جو) with (kon, کون).

(kea, کیا, what), (kon, کون, who) inflects in number and case. So a data type is defined the following way:

```haskell
data InterrPronForm = IntP Number Case
type InterrPron = InterrPronForm -> Str
```

and `pronInter` function is defined for their complete inflection.

(kəhan, کہان, where), (kəb, کب, when), (kiðhər, کہر, where), (kiyon, کيون, why), (kiyonkər, کيونکر, how) do not inflect at all. A function `pronInter1` is defined for them and a data type is defined the following way:

```haskell
data InterrPronForm1 = IntP1
type InterrPron1 = InterrPronForm1 -> Str
```

(kiṭna, کیتنا, how much) inflects in number, case and gender. A function `pronInterInfl_cng` is defined for it and a data type is defined the following way:

```haskell
data InterrPronForm2 = IntP2 Number Case Gender
type InterrPron2 = InterrPronForm2 -> Str
```

(kæsa, کیسا, which), (konsa, کونسا, which one) inflects in number and gender. `pronInterInfl_ng` function is defined for them.

4.5.1.5 Indefinite Pronoun

Indefinite pronouns refer to an unknown or undetermined person, place or thing. In Urdu there are following words used for it.

(koʔi, کوئی, someone), (kiʃi, کئی, someone) for people and (kutʃi, کچھ, some) for things. Two loan Arabic words (bɑ'z, بعض, some), (bɑ'ze, بعضس, some) and (flaŋ, فلان, someone) is also used as Indefinite Pronoun.
4.5.1.6 Repeated Pronoun

\(\text{kon-kon, who-who), (kufin}, \text{a little bit), (kufin-na-kufin}, \text{at least something), (ko?i-ko?i, very few), (ko?i-na-ko?i, some one), (kisi-kisi, very few), (kisi-na-kisi, some one) are repeated pronouns in Urdu. They do not inflect. A function } \text{pronRep function is defined for them.}\)

4.5.1.7 Relative pronoun

Relative pronouns in Urdu substitute the noun and also found in a sentence explaining something about the substituted noun.

For example:

\[\text{wo kÞab jo men-ne këridi } \text{hi}\]

That (Pron) book(noun) that(Pron) I(Erg. Pron) bought (verb Perf.) be(Past. Sg.)

The book (that) I bought

In above example \(džo, \text{jo}\) is relative pronoun. It inflects in number and case. \(\text{pron_jw function defines its inflection.}\)

- \(džehan, \text{where), (wëhan, where), (yëhan, here), (i gündar, here), (džighër, \text{then), (e inh, all), (džab, \text{second), (seb, \text{all}) inflects in case. A function } \text{pron_relative_c is defined for their inflection.}\)

- \(eas, \text{like this), (wëas, like that), (džësa, \text{is that), (itu, this much), (uëna, that much), (dusra, second) inflect in number and gender. A function } \text{pron_relative_ng is defined for them.}\)
• (ایا، آیا، either), (بته، بی، very), (هر، بر، all), (کهی، کنی، some), (کوی، same), (کل، net), (چند، some) do not inflect at all. A function `pron_relative_noInflect` is defined for them.

4.5.2 PostPositions, Particles and Numerals

We defined functions for the PostPositions, Particles and Numerals (one function for each) and their data types are defined in the following way:

```haskell
data PostPosForm = PostPosForm

type PostPosition = PostPosForm -> Str

data ParticleForm = ParticleForm

type Particle = ParticleForm -> Str

data NumeralForm = NumeralForm

type Numeral = NumeralForm -> Str
```

They do not inflect. Three interface functions are also defined for them so that words can be added afterwards.
Chapter 5

5 The Lexicon

5.1 The Extraction of Lexicon

A wide-coverage lexicon is a key part of any morphological implementation. Today, most of the lexicons are built manually, which is a very time consuming task. We aim to build a lexicon of Urdu automatically with minimal human effort. To extract a lexicon automatically, we use a tool named extract which is primarily developed for the morphologies developed in FM.

The tool combines regular expressions containing variables with propositional logic to form search patterns which identify lemmas tagged with their paradigm class (Forsberg, Hammarström & Ranta, 2005, page 1).

A morphology implementer provides a paradigm file and a corpus to the extract tool. A paradigm file consists of rules for all paradigms defined for that certain morphology. The extract tool reads the paradigms, search the corpus for those words that fulfill the definition of paradigms and extract them along with the name of the fulfilled paradigm.

For the extraction of Urdu lexicon, the first step was to collect a reasonable amount of Urdu Unicode text to make a corpus. We collected Urdu Unicode text from the following sources:

<table>
<thead>
<tr>
<th>Name/Nature of text</th>
<th>Amount</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>News</td>
<td>132 web pages</td>
<td>BBC Urdu service</td>
</tr>
<tr>
<td>Uranian named short stories of Urdu Literature –</td>
<td>139 web pages</td>
<td>Urdu book bank, (کتاب گھر)</td>
</tr>
<tr>
<td>Various Books on Urdu literature</td>
<td>445 web pages</td>
<td>Urdu book bank, (Urdu web, اردوویب)</td>
</tr>
</tbody>
</table>

Then all the html tags and other non-related information are thrown away by using the tools developed in this thesis work (section 3.5) and pure Unicode Urdu text is saved as text file. Then this Urdu Unicode text is converted into Roman Urdu by using transliteration tools (section 3.5).

Following are some statistics of the extracted Roman corpus:

Size of Corpus: 54.4MB
Tokens (words): 11,100,000 (eleven million)
Unique tokens (words): 49400
It is interesting to note the big difference between total words and unique words in the corpus. The unique words are considerably less than the total words. A statement that Urdu makes an extensive use of clitics, postpositions and auxiliaries, is verified by this observation.
We devised 23 rules (8 for verbs, 14 for nouns, 1 for adjectives) to make a paradigm file for Urdu.
For example the following paradigm is defined for singular masculine nouns that do not end with (əlf, \, a), (ʧhəlɪ hɛ, \, h), (‘æn, ع, e) and (әn, \, aN) (section 4.1.1.2):

```
regexp Not_aheaN = char* (char- ("a" | "h" | "e" | "aN");

paradigm n3[x:Not_aheaN] =
  x { (x & (x+"wN" | x+"w"))};
```

A regular expression Not_aheaN is defined. According to its definition, Not_aheaN can be any string except “a”, “h”, “e” and “aN”.

In the definition, we say that show all words (x) that:

- x does not end with “a”, “e”, “h”, “aN” and
- Has forms either x and x+”wN” or has forms x and x+”w”

These are the forms in which this noun groups actually inflects. The tool will extract all words from the corpus that fulfill these conditions. Since an interface function named n3 was defined for this noun group, we gave the same name to its paradigm definition so that we could get the output in a following format and we could save it directly in the lexicon (Urdu.lexicon):

```
n3  A2an
n3  AbXar
n3  Admy
...```

It will be interesting to look at the paradigm defined for irregular verbs that has direct & indirect causative forms. As these forms are irregular, we have to provide three basic forms to generate correct morphological inflection of such verbs. These three basic forms are the infinitive forms of intransitive/transitive, direct causative and indirect causative.

Keeping these factors in mind, its paradigm is defined as follows:

```
paradigm v4=
  x +"na" x+"ana" x+"wana"
  {x+"na" & (x+"ana" | x+"wana")};
```

This states that:

- Show the three word forms of a lemma, ending with “na”, “ana” and “wana” for a variable x fulfilling the condition below
- Either x ends with “na” & “ana” or x ends with “na” & “wana”
It results the output in a following format that could be saved directly in Urdu.lexicon:

v4 cTkna cTkana cTkwana
v4 caTna cTana cTwana
v4 ck|hna ck|hana ck|hwana

A complete paradigm file for Urdu can be seen in the source code named urdu.para.

Then the extract tool is applied on corpus along with the paradigm file resulting in an Urdu lexicon of 7000 words initially.

The result from extract tool could vary with respect to the following factors:

- The occurrence of misspellings, foreign words, numeric expressions, pronouns etc in the corpus
- The knowledge of the language with respect to its lexical distribution
- The level of strictness in the paradigm rules

Here strictness in the paradigm rules means a tighter definition of a paradigm rule by requiring more word forms

For example: the paradigm rule v4 could be made stricter in the following way:

```paradigm v4=
  x +"na" x+"ana" x+"wana"
  {x+"na" & x+"ana" & x+"wana"};
```

We require that only those words should be extracted that ends with “na”, “ana” and “wana”. The word forms of a lemma do not normally appear in a homogenous way in writings. Actually very few lemmas appear in all word forms. Therefore the tool will now extract fewer words for this paradigm but accuracy will definitely be increased as compared to the rule previously written for v4.

One can observe by looking at the paradigm file in the source code that we tried to make a fair balance between the strictness and the coverage. However, to be sure about the correctness of the lexicon, we manually re-checked the automatically built lexicon from word to word; and all incorrect words have been thrown away resulting in a lexicon of 4131 words generating 496976 word forms.
Chapter 6

6 Related Work

6.1 Morphology and Syntax Treatments

6.1.1 The Parallel Grammar Project (ParGram)

A large-scale on-going implementation of the Urdu grammar is the Parallel Grammar project (ParGram, 2002). The Parallel Grammar Project (ParGram henceforth) uses the XLE parser and grammar development platform for six languages: English, French, German, Japanese, Norwegian, and Urdu. The XLE parser is responsible to parse and generate the Lexical-Functional Grammar (LFG henceforth) formalism for all grammars. The aim of the project is to produce wide coverage grammars for above mentioned languages written collaboratively, within the linguistic framework of LFG and with a commonly-agreed-upon set of grammatical features (ParGram, 2002). The Urdu/Hindi morphology is currently under development as a part of ParGram (Urdu ParGram, 2003). This implementation is based on Xerox finite state technology and relies on Roman/ASCII transliteration. While future plans include the large-coverage extension of the morphology and the conversion from Roman/ASCII transliteration script to Perso-Arabic and Devanagari scripts. A number of publications for Urdu are written under this project.

6.1.2 The CRL Language Resources Project

An online Urdu-English Dictionary (lexicon) of 8k size including proper names and the Urdu Resource Package is available under the CLR Language Resource Project (Cowie & Abdelali, 2004). The Urdu Resource Package contains the following:

- A Morphological Analyzer
- A Text Preprocessor tool that recognizes dates
- A prototype Urdu-to-English word for word machine translation system
- 200 sentence parallel corpus of Urdu-English sentences
- Urdu-English bi-lingual lexicon which covers 200 sentence corpus

The Morphological Analyzer generates analyses for texts in Arabic, Persian and Urdu. The Morphological analyzer is written in ANSI C. From the inflection tables of the different paradigms of word classes (Noun, verb, Adjective and Adverb), morphology rules (sort of regular expressions) are automatically generated by tool Boas (Boas, 2000). Then these generated rules are further used by the MEAT morphological analyzer (MEAT, 1999) to analyze words. As stated by the authors, the text is initially analyzed by a tokenization step, which recognizes basic token types, inserts a representation in a hash table and produces a span descriptor for the token. After that all processing is carried out either on the hash table (for morphology and lexical lookup), or on the span descriptors for phrasal and pattern recognition.
6.1.3 The EMILLE Project

EMILLE (Enabling Minority Language Engineering) was a 3 year project at Lancaster University and Sheffield University. In this project 97 million word electronic corpus was generated for the South Asian languages (including Urdu). This corpus contains 200,000 words of parallel text in English, Bengali, Gujarati, Hindi, Punjabi and Urdu while the remainder is monolingual.

For Urdu, an automated part-of-speech tagger was further developed (Hardie 2003, 2004, 2005) which was then subsequently used to tag the whole Urdu corpus.

6.1.4 An Urdu ATN morphological parser

An Urdu Morphology Parser is developed by (Imran, 1997) as a Master thesis. This parser analyses the grammatical sequences of Romanized words in Urdu/Hindi. The ATN (Augmented Transition Network) used in this parser is based on GPARS system written in Lisp (Imran, 1997, page 3). As described by the author, the parser can parse simple as well as certain complex constructions like the relative clauses.

However the certain compound constructions like compound verbs (verbal + aux), the conjunct verbs (nominal + aux), compound postpositions, relative adverbs, adjectives, double relative pronouns have not been treated. Similarly the more complex constructions involving tenses, moods and aspect of the verb phrase are also not implemented in this work.

6.2 Transliteration Systems

6.2.1 Hindi Urdu Machine Transliteration System (HUMTS)

A notable transliteration system for Urdu and Hindi is M. G. Abbas Malik’s Hindi Urdu Machine Transliteration System (Malik, 2006). The Hindi Urdu Machine Transliteration System (HUMTS henceforth) takes Hindi or Urdu Unicode text and generates a common language (CL) which is based on ASCII/Roman. There can be found several major differences between Urdu and Hindi at phonological level (Naim, 1999, preface: page iii); e.g. having multiple letters to represent the same phonetic sound etc. This Transliteration system takes care of such special issues of both languages by applying Automatic Normalization Algorithm for Urdu and Hindi at the time of generating common language (CL).

The conversion rules, mapping and Automatic Normalization Algorithms for Urdu and Hindi are designed by using Xerox Finite state tools. A correct transliteration is performed by this system even if the diacritics/aarab are missing from the text. A very detailed analysis of Urdu, Hindi and the transliteration system is provided in the Master thesis report.

6.2.2 Hindi to Urdu Transliterator

Another transliterator for Urdu to Hindi and Hindi to Urdu is developed by (CRULP, Transliterator). The system is based on one to one string mapping of Hindi letters to their
corresponding Urdu letters. This system has many limitations such as it cannot handle the special issues of both languages as mentioned in section 6.2.1. However, in general, a number of keyboard layouts, fonts, number of publications for Urdu and other useful tools can be found from CRULP website.

6.3 Electronic Lexicons

The following notable, freely available electronic lexicons for Urdu are found on the web and are shown in the table below with necessary description:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Producer/Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Urdu Dictionary</td>
<td>A Comprehensive Urdu lexicon containing 56k word, of which 10k words have English glosses. Includes POS and Unicode based. Beta version is available for testing purposes.</td>
<td>Center for Research in Urdu Language Processing (CRULP, 2005)</td>
</tr>
<tr>
<td>The English Urdu Dictionary</td>
<td>A 35k Unicode based word lexicon with POS. A free web version of a commercial dictionary.</td>
<td>Pakistan Data Management Services software company (PDMS )</td>
</tr>
<tr>
<td>The English Urdu Dictionary</td>
<td>English to Urdu and Urdu to English lexicon of 5k words Urdu words are rendered in both roman and Urdu scripts.</td>
<td><a href="http://www.UrduWord.com">www.UrduWord.com</a></td>
</tr>
<tr>
<td>The Platts dictionary</td>
<td>An old dictionary which is a part of Digital Dictionaries of South Asia project at University of Chicago. Provides Devanagari and Roman transliteration. Perso-Arabic script not yet displaying.</td>
<td>(Platts dictionary, 1984)</td>
</tr>
<tr>
<td>The Shakespear dictionary</td>
<td>An old dictionary using Roman transliteration. A part of Digital Dictionaries of South Asia project at University of Chicago. The Devanagari and the Perso-Arabic scripts are not yet displaying</td>
<td>(Shakespear dictionary, 1834)</td>
</tr>
<tr>
<td>The English Urdu Dictionary</td>
<td>A lexicon of 25k words, using documented Roman transliteration.</td>
<td>(Siddiqi, 1997)</td>
</tr>
<tr>
<td>The Dictionary, Translation tool and Transliteration</td>
<td>English to Urdu Sentence Translator, English to Urdu and Urdu to Urdu Transliteration, a dictionary with 750 words and an Urdu Writer / Emailler is provided.</td>
<td>(ApniUrdu, 2002)</td>
</tr>
</tbody>
</table>
Chapter 7

7 Conclusion and Future Work

7.1 Conclusion

FM toolkit has proved to be a very good choice for implementing Urdu morphology. Haskell provided us a complete freedom for defining Urdu morphology with great ease. Dealing word classes and the parameters belonging to them as algebraic data types and the inflection tables (paradigms) for all word classes as finite functions satisfying the completeness makes this implementation more elegant, modular, extensible and reusable as compared to the existing implementations for Urdu morphology.

Further we successfully implemented the Urdu morphology with a considerate level of depth for every word class as described in previous chapters. The GUI tools, on screen Urdu Keyboard and transliteration scheme are proved to be very useful. We successfully extracted a lexicon of 4131 words generating 496976 word forms for Urdu, with a minimal human effort by an automatic lexicon extraction tool extract and the transliteration tools. All the implementation details are well documented in this thesis report. Further it is tried to make it as informative as possible with respect to the Urdu morphology and orthography so that interested audience could get start easily and contribute further.

7.2 Future work

A complete morphology for Urdu with all its small details is a big task. This project is available as an open source which we believe will help us getting comments, feedback and contribution from the audience. This project could be further enhanced with the following possible tasks/extensions:

- The lexicon could be extended further
- As the lexicon is extracted from the existing Urdu text available on the web, it may or may not contain the vowels and diacritic marks. An algorithm could be designed to apply the missing vowels and diacritic marks on the entries in the lexicon
- This system can be used equally for Hindi morphology provided by a transliteration scheme for Davanagari script
- The remaining less frequent group of words for nouns, adjectives etc could be defined in inflection engine.
- This system could be extended further from morphology to syntax and semantics specifically by using GF
7.3 Software Availability and Licensing

All the software libraries developed in this thesis work are freely available under GNU General Public License (GPL, 1991) and can be downloaded from FM home page (FM, 2004). These software libraries include Urdu morphology API, lexicon and the Unicode infrastructure mentioned in section 3.5.
8 References

(ApniUrdu, 2002): ApniUrdu website developed by Mohammad Nasim, Faisal Nasim & Raza Abbas
http://apniurdu.com/

(Beasley 1998): Kenneth R. Beesley. Romanization, Transcription and Transliteration
Xerox Arabic Morphological Analysis and Generation, 1998
http://www.xrce.xerox.com/competencies/content-analysis/arabic/info/romanization.html

(Boas. 2000): Boas: A Field Linguist in a Box developed by Sergei Nirenburg PI, Jim Cowie,
Marge McShane Victor Raskin, Kemal Oflazer. (Purdue, Bilkent and New Mexico State
University), 2000
http://www.ldc.upenn.edu/exploration/LSA/zacharski.html

(Butt, 2003): M. Butt. The morpheme that would’nt go away. Talk on Urdu causatives, held at
the Workshop on Pertinacity, Schloss Freudental, July 2002, updated handout from March

(Butt & King, 2004): M. Butt and T. H. King. The status of case. Clause Structure in South
Asian Languages, Series: Studies in Natural Language and Linguistic Theory, Vol. 61 Dayal,
Veneeta; Mahajan, Anoop (Eds.), 2004, VI
http://ling.uni-konstanz.de/pages/home/butt/butt-king.pdf

(Cowie & Abdelali, 2004): Jim Cowie and Ahmed Abdelali, Multi-Language Text Pre-
processor User Guide. CRL Language Resources Computing Research Laboraroy, New
Mexico State University, 2004
http://crl.nmsu.edu/Resources/lang_res/index.html

(CRULP, 2001): Nafees fonts, Center for research in Urdu language processing, National
University of Computer and Emerging Sciences (NUCES) Lahore. 2001,
http://www.crulp.org

(CRULP, 2005): The Urdu Dictionary, Center for research in Urdu language processing,
National University of Computer and Emerging Sciences (NUCES) Lahore, Beta as of 18 Aug
2005
http://www.crulp.org/oud

(CRULP, Transliterator): Hindi to Urdu Transliterator, , Center for research in Urdu language
processing, National University of Computer and Emerging Sciences (NUCES) Lahore
http://www.crulp.org/h2utransliterator.html
(CRULP, Urdu phonetic keyboard): Urdu phonetic keyboard, Center for research in Urdu language processing, National University of Computer and Emerging Sciences (NUCES) Lahore
http://www.crulp.org/Downloads/Phonetic%20KeyBoard.pdf

http://www.cs.chalmers.se/~markus/FM/


http://www.fsf.org/licenses/gpl.txt


http://eprints.lancs.ac.uk/103/01/cl03_urdu.pdf


(Haskell Introduction) For general information regarding Functional Programming following is a link, http://www.haskell.org/aboutHaskell.html

(Huet, 2000): Gérard Huet's Sanskrit Site, Program and documentation, 2000
http://sanskrit.inria.fr/

(Huet, 2002): Gérard Huet, the Zen Computational Linguistics Toolkit, 2002
http://sanskrit.inria.fr/huet/ZEN/index.html
(ICU4J 3.4, 2006): International Components for Unicode for Java. Version 3.6, an open source project by IBM Corporation
http://icu.sourceforge.net/userguide/intro.html

http://faculty.mdc.edu/simran/urduParser2.doc

http://www.puran.info/HUMTS/HUMTS.htm

(MEAT, 1999): The Multilingual Environment for Advanced Translations, developed at Computing Research Laboratory 1999-2000
http://crl.nmsu.edu/~ahmed/Meat/index.html

(Naim, 1999): C. M. Naim, Introductory Urdu volume 1, Revised 3rd edition, South Asian Language & Area Center, University of Chicago, 1999

http://en.wikipedia.org/wiki/Nastaleeq

http://www2.parc.com/isl/groups/nltt/pargram/buttetal-coling02.pdf

(PDMS): Pakistan Data Management Services
http://urduseek.com/


http://dsal.uchicago.edu/dictionaries/platts/


(Siddiqi, 1971): “dʒɑmeʊl-qwɑʔid - جامع القواعد” by Dr. Abullais Siddiqi (ڈاکٹر ابول الیس صدیقی), Head of the Department of Urdu, Karachi University, Markazi Urdu Board, Lahore, 1971

http://biphost.spray.se/tracker/dict/index.html

http://dsal.uchicago.edu/dictionaries/shakespear/

http://ling.uni-konstanz.de/pages/home/butt/pargram/index.html


Note: All the web addresses given above were last accessed in September 2006.
# Appendix A - Manual for lexicographers

## 9.1 The Transliteration Scheme

### The Letters:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>!t</td>
<td>ت</td>
</tr>
<tr>
<td>a^</td>
<td>أ</td>
</tr>
<tr>
<td>w^</td>
<td>و</td>
</tr>
<tr>
<td>y^</td>
<td>ي</td>
</tr>
<tr>
<td>n</td>
<td>ن</td>
</tr>
<tr>
<td>f</td>
<td>ف</td>
</tr>
<tr>
<td>q</td>
<td>ق</td>
</tr>
<tr>
<td>k</td>
<td>ك</td>
</tr>
<tr>
<td>h</td>
<td>ه</td>
</tr>
<tr>
<td>g</td>
<td>غ</td>
</tr>
<tr>
<td>x</td>
<td>خ</td>
</tr>
<tr>
<td>c</td>
<td>چ</td>
</tr>
<tr>
<td>A</td>
<td>آ</td>
</tr>
<tr>
<td>m</td>
<td>م</td>
</tr>
<tr>
<td>&amp;</td>
<td>ء</td>
</tr>
<tr>
<td>h^</td>
<td>ه</td>
</tr>
<tr>
<td>l</td>
<td>ل</td>
</tr>
<tr>
<td>t</td>
<td>ت</td>
</tr>
<tr>
<td>d</td>
<td>د</td>
</tr>
<tr>
<td>p</td>
<td>پ</td>
</tr>
<tr>
<td>h</td>
<td>ه</td>
</tr>
<tr>
<td>j</td>
<td>ج</td>
</tr>
<tr>
<td>C</td>
<td>ث</td>
</tr>
<tr>
<td>E</td>
<td>س</td>
</tr>
<tr>
<td>Z</td>
<td>ذ</td>
</tr>
<tr>
<td>R</td>
<td>ر</td>
</tr>
</tbody>
</table>

### The Vowels (Alephab / Harrukh):

<table>
<thead>
<tr>
<th>Vowel</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>[i]</td>
<td>kʰer-i-zer</td>
</tr>
<tr>
<td>(A)</td>
<td>do-zaber</td>
</tr>
<tr>
<td>(I)</td>
<td>do-zer</td>
</tr>
<tr>
<td>(O)</td>
<td>do-pef</td>
</tr>
<tr>
<td>(a)</td>
<td>zaber</td>
</tr>
<tr>
<td>(i)</td>
<td>zer</td>
</tr>
<tr>
<td>(o)</td>
<td>pef</td>
</tr>
<tr>
<td>[a]</td>
<td>kʰeri-zaber</td>
</tr>
</tbody>
</table>

### The Numerals:

<table>
<thead>
<tr>
<th>Numeral</th>
<th>Arabic Numeral</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 _ur</td>
<td>۶</td>
</tr>
<tr>
<td>2 _ur</td>
<td>۲</td>
</tr>
<tr>
<td>3 _ur</td>
<td>۳</td>
</tr>
<tr>
<td>4 _ur</td>
<td>۴</td>
</tr>
<tr>
<td>5 _ur</td>
<td>۵</td>
</tr>
<tr>
<td>6 _ur</td>
<td>۶</td>
</tr>
<tr>
<td>7 _ur</td>
<td>۷</td>
</tr>
<tr>
<td>8 _ur</td>
<td>۸</td>
</tr>
<tr>
<td>9 _ur</td>
<td>۹</td>
</tr>
</tbody>
</table>

### The Special symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>؟</td>
</tr>
<tr>
<td>[RZ]</td>
<td>RZ</td>
</tr>
<tr>
<td>[SLM]</td>
<td>SLM</td>
</tr>
<tr>
<td>[ALLAH]</td>
<td>ALLAH</td>
</tr>
<tr>
<td>[RA]</td>
<td>RA</td>
</tr>
<tr>
<td>[AS]</td>
<td>AS</td>
</tr>
<tr>
<td>[SAW]</td>
<td>SAW</td>
</tr>
</tbody>
</table>
9.2 Word Classes:

Nouns
n1 l(a)R'ka ِ لازکا Singular masculine nouns ending with (ءلِف، ۱، a)، (تُؤِثی he، ۳، h) and ('اًي، ع، e)
n2 k(o)nw'an گُنوان Singular masculine nouns ending with (اًي، an، aN)
n3 m(a)r'd ۱ مرد Singular masculine nouns not ending with (ءلِف، ۱، a)، (تُؤِثی he، ۳، h)، ('اًي، ع، e) and (اًي، an، aN)
n4 k(o)r'sy کُسی Singular feminine nouns ending with (تُؤِثی ye، ۱， y)
n5 maN مان Singular feminine nouns ending with (ءلِف، ۱، a)، (اًي، an، aN)، (وَن، wN)
n6 g(o)R'ya گُریا Singular feminine nouns ending with (یا، ی، ya)
n7 H(a)ya حیا Singular loan feminine nouns ending with (یا، ی، ya)، exception from the above rule
n8 KwX'b(o)w خوشبو Singular feminine nouns ending with (وَو، w)
n9 k(i)tab کتاب Singular feminine nouns not ending with (ءلِف، ۱، a)، (وَن، w)، (وَن، wN)

Loan Arabic nouns:

n10 a(i)H'san اَحْسان Singular masculine nouns ending with (وَن، n، ن)، (آ، ر، ar)
n11 AK(i)r آَخ Singular masculine nouns starting with (ءلِف، ۱، a)، (ءلِف mِدَدَة، ۱، A) and ending with (ر، r، re)
n12 n(a)dam(a)t نَدَامَت Singular feminine nouns ending with (ت، ث، t)

Lean Persian nouns:

n13 Al(o)w أَل Singular masculine nouns ending with (وَو، w)
n14 b|haw^ بِهَاوُ Singular masculine nouns ending with (وَو، w)، (هِمْزَة، ِ، w^) or the nouns with no inflection
Names:

n15 h(o)may(a)wHumayoun مـُــُـُمـُـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُمـُـُـُم~
Masculine names

n16 nad(i)y(a)hNadia ناـدـىـہ Feminine names

n17 pak(i)s'tanPakistan پاکِسَـٰـتَـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰ~ Name of places

Verbs

The verbs have only basic stem form (intransitive/transitive etc), while direct causative & indirect causative cannot be made

v1 lana lana لاـنَا

The direct & indirect causative can be made from the basic stem form (intransitive/transitive etc), made by rules:

v2 b(a)yT'|hna betyna پْـْہـِـَّـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰ~

v3 n(i)k(a)l'na nikielna، سَـْـظْـہـِـَّـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰـٰ~

v4 m(i)l'na m(l)ana m(l)'wana All remaining and irregular verbs

Only direct causative can be made from the basic stem form (intransitive/transitive etc):

v5 shna sharna سـْـہـِـَّـٰـٰـٰ~

Only indirect causative can be made from the basic stem form (intransitive/transitive etc):

v6 bycna bycwana بـِـچـِـَّـٰـٰـٰ~
### Adjectives

<table>
<thead>
<tr>
<th>Adj 1</th>
<th>Inflections</th>
</tr>
</thead>
<tbody>
<tr>
<td>b(a)d</td>
<td>Inflects only in Persian way:</td>
</tr>
<tr>
<td></td>
<td>Without spaces: bəd̪, bəd̪t̪ər, bəd̪t̪ərìn</td>
</tr>
<tr>
<td></td>
<td>With spaces: bəd̪, bəd̪-t̪ər, bəd̪-t̪ərìn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adj 2</th>
<th>Inflection in Number and gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>b(o)ra</td>
<td>Both Urdu and Persian way of inflection:</td>
</tr>
<tr>
<td></td>
<td>Without spaces: bʊrɑ, bohət̪-bʊrɑ</td>
</tr>
<tr>
<td></td>
<td>With spaces: bʊrɑ, bohət̪-bʊrɑ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adj 3</th>
<th>Inflection in Number and gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>n(i)yla</td>
<td>Inflection in Number and gender</td>
</tr>
</tbody>
</table>

### Adverbs

<table>
<thead>
<tr>
<th>Adv 1</th>
<th>All adverbs will be added here</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(a)myX(a)</td>
<td>hmeʃə, همیشه</td>
</tr>
</tbody>
</table>

### The Closed classes

<table>
<thead>
<tr>
<th>Pos</th>
<th>Postpositions</th>
</tr>
</thead>
<tbody>
<tr>
<td>postpos</td>
<td>ka</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part</th>
<th>Participles</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td>تو</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Num</th>
<th>Numerals</th>
</tr>
</thead>
<tbody>
<tr>
<td>num ayk</td>
<td>ایک</td>
</tr>
</tbody>
</table>
### Appendix B - Verb conjugations

#### 10.1 Conjugation for auxiliary (honә, hwna, be)

<table>
<thead>
<tr>
<th>Tense</th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Past</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ٙәٗ, تها</td>
<td>ٙәٗ, تها</td>
<td>ٙәٗ, تها</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>ٙiٗ, تهيٗ</td>
<td>ٙiٗ, تهيٗ</td>
<td>ٙiٗ, تهيٗ, تهيٗ</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>تٗ</td>
<td>تٗ</td>
<td>تٗ</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>تٗ</td>
<td>تٗ</td>
<td>تٗ</td>
</tr>
<tr>
<td>Present</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ہون</td>
<td>بون</td>
<td>ہون</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>ہٗ, ب٣ٗ</td>
<td>ہٗ, ب٣ٗ</td>
<td>ہٗ, ب٣ٗ</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>ب٣ٗ</td>
<td>ب٣ٗ</td>
<td>ب٣ٗ</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>ب٣ٗ</td>
<td>ب٣ٗ</td>
<td>ب٣ٗ</td>
</tr>
<tr>
<td>Future</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ہون-گاک</td>
<td>بون-گاک</td>
<td>ہون-گاک</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>ہٗ-گی</td>
<td>ب٣ٗ-گی</td>
<td>ب٣ٗ-گی</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>ب٣ٗ-گی</td>
<td>ب٣ٗ-گی</td>
<td>ب٣ٗ-گی</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>ب٣ٗ-گی</td>
<td>ب٣ٗ-گی</td>
<td>ب٣ٗ-گی</td>
</tr>
<tr>
<td>Subjunctive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ہون, بون</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
<td>ب٣ٗ, ب٣ٗ</td>
</tr>
<tr>
<td>Imperative- Same as Subjunctive but First person does not exist for it</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Imperfective

<table>
<thead>
<tr>
<th>Tense</th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>ہٗ</td>
<td>ب٣ٗ</td>
<td>ب٣ٗ</td>
</tr>
</tbody>
</table>
### 10.2 Conjugation for Single-word-analysis of verb

The conjugation table for verb (lana, لانا, lana, to bring):

<table>
<thead>
<tr>
<th>Pl. Masc</th>
<th>Pl. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td>hoːe</td>
<td>hoːi</td>
</tr>
</tbody>
</table>

#### Perfective

<table>
<thead>
<tr>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casuа</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>hoːa</td>
<td>hoːe</td>
</tr>
<tr>
<td>Sg Fem</td>
<td>hoːi</td>
<td>hoːe</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>hoːe</td>
<td>hoːi</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>hoːi</td>
<td>hoːi</td>
</tr>
</tbody>
</table>

#### Imperfective

<table>
<thead>
<tr>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casuа</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>laʔa</td>
<td>laʔe</td>
</tr>
<tr>
<td>Sg Fem</td>
<td>laʔi</td>
<td>laʔi</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>laʔe</td>
<td>laʔe</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>laʔi</td>
<td>laʔi</td>
</tr>
</tbody>
</table>
10.3 Conjugation for Combination-analysis of a verb

10.3.1 Past Tense Conjugation

<table>
<thead>
<tr>
<th>Sg Fem</th>
<th>laṭi</th>
<th>laτi</th>
<th>laτi</th>
<th>laτi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pl. Masc</td>
<td>laτे</td>
<td>laτे</td>
<td>laτे</td>
<td>laτे</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>laτינ</td>
<td>laτین</td>
<td>laτین</td>
<td>laτین</td>
</tr>
</tbody>
</table>

### Indefinite

<table>
<thead>
<tr>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Near</td>
<td>Distant</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sg. Masc</th>
<th>лаиа лаи лаи лаиа</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sg Fem</td>
<td>ла?и ла?и ла?и</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>ла?е ла?е</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>ла?и</td>
</tr>
</tbody>
</table>

### Imperfective

<table>
<thead>
<tr>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Near</td>
<td>Distant</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sg. Masc</th>
<th>лаτа-лл̱а лаτа-лл̱а</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sg Fem</td>
<td>лаτи-лл̱и лаτи-лл̱и</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>лаτе-лл̱е</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>лаτи-лл̱и</td>
</tr>
</tbody>
</table>

### Imperfective Continuous

<table>
<thead>
<tr>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Near</td>
<td>Distant</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sg. Masc</th>
<th>ла-ра-лл̱а ла-ра-лл̱а</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sg Fem</td>
<td>ла-ри-лл̱и ла-ри-лл̱и</td>
</tr>
</tbody>
</table>

95
### Imperfective Habitual

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Masc</strong></td>
<td>laya-kərtə-<em>[^h]a</em></td>
<td>laya-kərtə-<em>[^h]a</em></td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Fem</strong></td>
<td>laya-kərtə-<em>[^h]i</em></td>
<td>laya-kərtə-<em>[^h]i</em></td>
</tr>
<tr>
<td><strong>Pl.</strong></td>
<td><strong>Masc</strong></td>
<td>laya-kərtə-<em>[^h]a</em></td>
<td>laya-kərtə-<em>[^h]a</em></td>
</tr>
<tr>
<td><strong>Pl.</strong></td>
<td><strong>Fem</strong></td>
<td>laya-kərtə-<em>[^h]i</em></td>
<td>laya-kərtə-<em>[^h]i</em></td>
</tr>
</tbody>
</table>

### Imperfective Habitual Continuous

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Masc</strong></td>
<td>la[t-]ra[^a]</td>
<td>la[t-]r[^h]e</td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Fem</strong></td>
<td>la[t-]r[^a]i</td>
<td>la[t-]r[^h]i</td>
</tr>
<tr>
<td><strong>Pl.</strong></td>
<td><strong>Masc</strong></td>
<td>la[t-]r[^a]</td>
<td>la[t-]r[^a]</td>
</tr>
<tr>
<td><strong>Pl.</strong></td>
<td><strong>Fem</strong></td>
<td>la[t-]r[^a]i</td>
<td>la[t-]r[^a]i</td>
</tr>
</tbody>
</table>

### Perfective Immediate

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Masc</strong></td>
<td>la[t-f]uka-<em>[^h]a</em></td>
<td>la[t-f]uka-<em>[^h]a</em></td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Fem</strong></td>
<td>la[t-f]uka-<em>[^h]i</em></td>
<td>la[t-f]uka-<em>[^h]i</em></td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Masc</strong></td>
<td>la[t-f]uka-<em>[^h]a</em></td>
<td>la[t-f]uka-<em>[^h]a</em></td>
</tr>
<tr>
<td><strong>Sg.</strong></td>
<td><strong>Fem</strong></td>
<td>la[t-f]uka-<em>[^h]i</em></td>
<td>la[t-f]uka-<em>[^h]i</em></td>
</tr>
</tbody>
</table>

[^h]: Plural
[^i]: Singular
<table>
<thead>
<tr>
<th></th>
<th>Pl. Masc</th>
<th>Pl. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Person</strong></td>
<td>la-ʃuken</td>
<td>la-ʃuki</td>
</tr>
<tr>
<td><strong>Second Person</strong></td>
<td>ho</td>
<td>ho</td>
</tr>
<tr>
<td><strong>Third Person</strong></td>
<td>hen</td>
<td>hen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pl. Masc</th>
<th>Pl. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Person</strong></td>
<td>la-ʃuken</td>
<td>la-ʃuki</td>
</tr>
<tr>
<td><strong>Second Person</strong></td>
<td>ho</td>
<td>ho</td>
</tr>
<tr>
<td><strong>Third Person</strong></td>
<td>hen</td>
<td>hen</td>
</tr>
</tbody>
</table>

**Presumptive**

<table>
<thead>
<tr>
<th></th>
<th>Sg. Masc</th>
<th>Sg. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Person</strong></td>
<td>laya-huŋ-ga</td>
<td>laʔi-huŋ-gi</td>
</tr>
<tr>
<td><strong>Second Person</strong></td>
<td>laya-ho-ga</td>
<td>laʔi-ho-gi</td>
</tr>
<tr>
<td><strong>Third Person</strong></td>
<td>laʔe-ho-he</td>
<td>laʔe-ho-he</td>
</tr>
</tbody>
</table>

**Conditional**

<table>
<thead>
<tr>
<th></th>
<th>Sg. Masc</th>
<th>Sg. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Person</strong></td>
<td>laʔa</td>
<td>laʔi</td>
</tr>
</tbody>
</table>
### Present Tense Conjugation

**10.3.2 Present Tense Conjugation**

<table>
<thead>
<tr>
<th></th>
<th>Pl. Masc</th>
<th>Pl. Fem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sg. Masc</strong></td>
<td>laʔe-hoṭ</td>
<td>laʔi-hoṭ</td>
</tr>
<tr>
<td><strong>Sg. Fem</strong></td>
<td>laʔe-hoṭ</td>
<td>laʔi-hoṭ</td>
</tr>
</tbody>
</table>

#### Indefinite

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sg. Masc</strong></td>
<td>laʔa-hun</td>
<td>laʔa-he</td>
<td>laʔa-ho</td>
</tr>
<tr>
<td><strong>Sg. Fem</strong></td>
<td>laʔa-hun</td>
<td>laʔa-he</td>
<td>laʔa-ho</td>
</tr>
<tr>
<td><strong>Pl. Masc</strong></td>
<td>laʔa-hun</td>
<td>laʔa-he</td>
<td>laʔa-ho</td>
</tr>
<tr>
<td><strong>Pl. Fem</strong></td>
<td>laʔa-hun</td>
<td>laʔa-he</td>
<td>laʔa-ho</td>
</tr>
<tr>
<td></td>
<td>First Person</td>
<td>Second Person</td>
<td>Third Person</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>la-rha-hun</td>
<td>la-rha-he</td>
<td>la-rha-ho</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>la-rhi-hun</td>
<td>la-rhi-he</td>
<td>la-rhi-ho</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>la-rhe-hen</td>
<td>la-rhe-ho</td>
<td>la-rhe-ho</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>la-rhi-hen</td>
<td>la-rhi-ho</td>
<td>la-rhi-ho</td>
</tr>
</tbody>
</table>

**Imperative**

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td></td>
<td>la</td>
<td>la?o</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>Does not exist</td>
<td></td>
<td>la?e</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>la?o</td>
<td></td>
<td>la?e</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>la?o</td>
<td></td>
<td>la?e</td>
</tr>
</tbody>
</table>
### 10.3.3 Future Tense Conjugation

#### Indefinite

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>laʔun-ga</td>
<td>laʔ-e-ga</td>
<td>laʔ-o-ge</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>laʔun-gi</td>
<td>laʔ-e-gi</td>
<td>laʔ-o-gi</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>laʔen-ge</td>
<td>laʔ-o-ge</td>
<td>laʔ-o-ge</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>laʔen-gi</td>
<td>laʔ-o-gi</td>
<td>laʔ-o-gi</td>
</tr>
</tbody>
</table>

#### Imperfective Continuous

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>laʔ̱-ṟhun-ga</td>
<td>laʔ̱-ṟhe-ga</td>
<td>laʔ̱-ṟhe-ga</td>
</tr>
<tr>
<td>Sg. Fem</td>
<td>laʔ̱-ṟhun-gi</td>
<td>laʔ̱-ṟhe-gi</td>
<td>laʔ̱-ṟhe-gi</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>laʔ̱-ṟhen-ge</td>
<td>laʔ̱-ṟhe-ga</td>
<td>laʔ̱-ṟhe-ga</td>
</tr>
<tr>
<td>Pl. Fem</td>
<td>laʔ̱-ṟhen-gi</td>
<td>laʔ̱-ṟhe-gi</td>
<td>laʔ̱-ṟhe-gi</td>
</tr>
</tbody>
</table>

#### Presumptive

<table>
<thead>
<tr>
<th></th>
<th>First Person</th>
<th>Second Person</th>
<th>Third Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Casual</td>
<td>Familiar</td>
<td>Respect</td>
</tr>
<tr>
<td>Sg. Masc</td>
<td>laʔa-hun-ga</td>
<td>laʔa-ho-ga</td>
<td>laʔe-ho-ga</td>
</tr>
<tr>
<td></td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Sg Fem</td>
<td>laṭи-hun-gi</td>
<td>laṭи-ho-gi</td>
<td>laṭи-ho-gи</td>
</tr>
<tr>
<td></td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
</tr>
<tr>
<td>Pl. Masc</td>
<td>laṭе-hon-ge</td>
<td>laṭе-hо-ge</td>
<td>laṭе-hо-ge</td>
</tr>
<tr>
<td></td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
</tr>
<tr>
<td></td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
<td>laṭи-bоп-а</td>
</tr>
</tbody>
</table>
## 11 Appendix C - Inflection Tables

### 11.1 Personal Pronouns:

<table>
<thead>
<tr>
<th></th>
<th>Nom</th>
<th>Erg</th>
<th>Acc</th>
<th>Dat</th>
<th>Inst</th>
<th>Gen</th>
<th>Loc</th>
<th>Voc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Sg</strong></td>
<td>men</td>
<td>men-</td>
<td>meɳ</td>
<td>meɳ- ko / se</td>
<td>meɳ</td>
<td>meɳ- ko / se</td>
<td>mera</td>
<td>mera</td>
</tr>
<tr>
<td><strong>2 Sg</strong></td>
<td>to</td>
<td>to-</td>
<td>to-</td>
<td>tudʒh- ko / se</td>
<td>tudʒh</td>
<td>tudʒh- ko / se</td>
<td>tumhara</td>
<td>tumhara</td>
</tr>
</tbody>
</table>

**Very Casual**

<table>
<thead>
<tr>
<th></th>
<th>Nom</th>
<th>Erg</th>
<th>Acc</th>
<th>Dat</th>
<th>Inst</th>
<th>Gen</th>
<th>Loc</th>
<th>Voc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Sg</strong></td>
<td>men</td>
<td>men-</td>
<td>meɳ</td>
<td>meɳ- ko / se</td>
<td>meɳ</td>
<td>meɳ- ko / se</td>
<td>mera</td>
<td>mera</td>
</tr>
<tr>
<td><strong>2 Sg</strong></td>
<td>to</td>
<td>to-</td>
<td>to-</td>
<td>tudʒh- ko / se</td>
<td>tudʒh</td>
<td>tudʒh- ko / se</td>
<td>tumhara</td>
<td>tumhara</td>
</tr>
<tr>
<td>2 Sg Familia r</td>
<td>tum</td>
<td>tum-ne</td>
<td>tum-ko / se</td>
<td>tumhe</td>
<td>tum-se</td>
<td>same as above</td>
<td>tum-men / per / țek / țalək</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>--------</td>
<td>---------------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>2 Sg Respect</td>
<td>ap</td>
<td>ap-ne</td>
<td>ap-ko / ke / se</td>
<td>ap-ke</td>
<td>ap-se</td>
<td>ap-ko / ke</td>
<td>ap-men / per / țek / țalək</td>
<td></td>
</tr>
<tr>
<td>2 Pl Very Casual</td>
<td>tum</td>
<td>tum-ne</td>
<td>tum-ko / se</td>
<td>tumhe</td>
<td>tum-se</td>
<td>tum-men / per / țek / țalək</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Pl Familia r</td>
<td>tum</td>
<td>tum-ne</td>
<td>tum-ko / se</td>
<td>tumhe</td>
<td>tum-se</td>
<td>tum-men / per / țek / țalək</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Sg Near</td>
<td>ye</td>
<td>is-ne</td>
<td>is-ko / ke / se</td>
<td>is-se</td>
<td>is-se</td>
<td>is-ko / ki / ke</td>
<td>is-men / per / țek / țalək</td>
<td></td>
</tr>
</tbody>
</table>

- **2 Sg Familia r**: *tum* (تم), *tum-ne* (تم نے), *tum-ko / se* (تم کو / تم سے), *tumhe* (تم ہے), *tum-se* (تم سے), *same as above* (تم میں تھا / تم پر / تم تک / تم تلک)
- **2 Sg Respect**: *ap* (آپ), *ap-ne* (آپ نے), *ap-ko / ke / se* (آپ کو / آپ سے), *ap-ke* (آپ کے), *ap-se* (آپ سے), *ap-men / per / țek / țalək* (آپ مین / آپ پر / آپ تک / آپ تلک)
- **3 Sg Near**: *ye* (یہ), *is-ne* (اس نے), *is-ko / ke / se* (اس کو / اس سے), *is-se* (اس سے), *is-ko / ki / ke* (اس کی / اس کے), *is-men / per / țek / țalək* (اس مین / اس پر / اس تک / اس تلک)

- **None**: *none*
<table>
<thead>
<tr>
<th>3 Sg</th>
<th>3 Pl</th>
<th>3 Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distant</td>
<td>Near</td>
<td>Distant</td>
</tr>
<tr>
<td>wo 🗣️</td>
<td>ye 🗣️</td>
<td>wo 🗣️</td>
</tr>
<tr>
<td>us-ne 🗣️</td>
<td>in-ne 🗣️</td>
<td>un-ne 🗣️</td>
</tr>
<tr>
<td>us-ko/ke/se use 🗣️</td>
<td>in-ko / ke / se inhenŋ</td>
<td>un-ko / ke / se unhenŋ</td>
</tr>
<tr>
<td>us-ko/ke use 🗣️</td>
<td>in-ko / ke inhenŋ</td>
<td>un-ko / ke unhenŋ</td>
</tr>
<tr>
<td>us-se 🗣️</td>
<td>in-se 🗣️</td>
<td>un-se 🗣️</td>
</tr>
<tr>
<td>us-men/pər/ɪək/ɪəl ək 🗣️</td>
<td>in-men / pər / ɪək / ɪəl ək 🗣️</td>
<td>un-men / pər / ɪək / ɪəl ək 🗣️</td>
</tr>
</tbody>
</table>

none

none

none
### Relative pronoun – چو:

<table>
<thead>
<tr>
<th></th>
<th>Nom</th>
<th>Erg</th>
<th>Acc</th>
<th>Dat</th>
<th>Inst</th>
<th>Gen</th>
<th>Loc</th>
<th>Voc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Sg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Pl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dʒis-ko / ke</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-men / per / ṭək / ṭələk</td>
</tr>
<tr>
<td>2 Sg</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒis-co / ke / se</td>
<td>dʒis-ke / se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>None</td>
</tr>
<tr>
<td>2 Pl</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒi-n-co / ke / se</td>
<td>dʒi-n-ke / se</td>
<td>dʒi-n-se</td>
<td>dʒi-n-se</td>
<td>dʒi-n-se</td>
<td>dʒi-n-se</td>
<td>dʒi-n-se</td>
<td>None</td>
</tr>
<tr>
<td>3 Sg</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>dʒo</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>dʒo-ne</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>dʒis-co / ke / se</td>
<td>dʒis-ke / se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>dʒis-se</td>
<td>None</td>
</tr>
<tr>
<td>3 Pl</td>
<td>30جو</td>
<td>30نجین-نا</td>
<td>30نجین-کو</td>
<td>30نجین-کو / ke / se</td>
<td>30نجین-کو / ke</td>
<td>30نجین-کو / ke</td>
<td>30نجین-کو / ke</td>
<td>30نجین-کو / ke</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن کو</td>
<td>جن کو</td>
<td>جن کو</td>
<td>جن کو</td>
<td>جن کو</td>
<td>جن کو</td>
<td>جن کو</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن کی</td>
<td>جن کی</td>
<td>جن کی</td>
<td>جن کی</td>
<td>جن کی</td>
<td>جن کی</td>
<td>جن کی</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
<tr>
<td></td>
<td></td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
<td>جن سے</td>
</tr>
</tbody>
</table>