An Open Source Urdu Resource Grammar

Shafqat M Virk
Department of Applied IT
University of Gothenburg
virk@chalmers.se

Muhammad Humayoun
Laboratory of Mathmatics
University of Savoie
mhuma@univ-savoie.fr

Aarne Ranta
Department of CS & Eng
University of Gothenburg
aarne@chalmers.se

Abstract

We develop a grammar for Urdu in Grammatical Framework (GF). GF is a programming language for defining multilingual grammar applications. GF resource grammar library currently supports 16 languages. These grammars follow an Interlingua approach and consist of morphology and syntax modules that cover a wide range of features of a language. In this paper we explore different syntactic features of the Urdu language, and show how to fit them in the multilingual framework of GF. We also discuss how we cover some of the distinguishing features of Urdu such as, ergativity in verb agreement (see Sec 4.2). The main purpose of GF resource grammar library is to provide an easy way to write natural language applications without knowing the details of syntax, morphology and lexicon. To demonstrate it, we use Urdu resource grammar to add support for Urdu in the work reported in (Angelov and Ranta, 2010) which is an implementation of Attempto (Attempto 2008) in GF.

1. Introduction

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

We develop a grammar for Urdu that addresses problems related to automated text translation using an Interlingua approach and provide a way to precisely translate text. This is described in Section 2. Then we describe different levels of grammar development including morphology (Section 3) and syntax (Section 4). In Section 6, we discuss an application in which a semantics-driven translation system is built upon these components.

2. GF (Grammatical Framework)

GF (Grammatical Framework, Ranta 2004) is a tool for working with grammars, implementing a programming language for writing grammars which in term is based on a mathematical theory about languages and grammars[1]. Many multilingual dialog and text generation applications have been built using GF. GF grammars have two levels *the abstract* and *the concrete syntax*[2]. The abstract syntax is language independent and is common to all languages in GF grammar library. It is based on common syntactic or semantic constructions, which work for all the involved languages on an appropriate level of abstraction. The concrete syntax is language dependent and defines a mapping from abstract to actual textual representation in a specific language[2]. GF uses the term 'category' to model different parts of speech (e.g verbs, nouns adjectives etc.). An abstract syntax defines a set of categories, as well as a set of tree building functions. Concrete syntax contains rules telling how these trees are linearized. Separating the tree building rules (abstract syntax) from linearization rules (concrete syntax) makes it possible to have multiple concrete syntaxes for one abstract. This

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

[2] In given example code 'fun' and 'cat' belongs to abstract syntax, 'lin' and 'lincat' belongs to concrete syntax

makes it possible to parse text in one language and translate it to multiple languages.

Grammars in GF can be roughly classified into two kinds: resource grammars and application grammars. Resource grammars are general purpose grammars (Ranta, 2009a) that try to cover the general aspects of a language linguistically and whose abstract syntax encodes syntactic structures. Application grammars, on the other hand, encode semantic structures, but in order to be accurate they are typically limited to specific domains. However, they are not written from scratch for each domain, but they use resource grammars as libraries (Ranta 2009b).

Previously GF had resource grammars for 16 languages: English, Italian, Spanish, French, Catalan, Swedish, Norwegian, Danish, Finish, Russian, Bulgarian, German, Interlingua (an artificial language), Polish, Romanian and Dutch. Most of these languages are European languages. We developed resource grammar for Urdu making it the 17th in total and the first south Asian language. Resource grammars for several other languages (e.g. Arabic, Turkish, Persian, Maltese and Swahili) are under construction.

3. Morphology

In GF resource grammars a test lexicon of 350 words is provided for each language. These words are built through lexical functions. The rules for defining Urdu morphology are borrowed from (Humayoun et el., 2006), in which Urdu morphology was developed in the Functional Morphology toolkit (Forsberg and Ranta, 2004). Although it is possible to automatically generate equivalent GF code from it, we wrote the rules of morphology from scratch in GF, to receive better abstractions than are possible in generated code. Furthermore, we extend this work by including compound words. However, the details of morphology are beyond the scope of this paper, and its focus is on syntax.

4. Syntax

While morphological analysis deals with the formation and inflection of individual words, syntax shows how these words (parts of speech) are grouped together to build well formed phrases. In this section we show how this works and is implemented for Urdu.

4.1 Noun Phrases (NP)

When nouns are to be used in sentences as part of speech, then there are several linguistic details which need to be considered. For example other words can modify a noun, and nouns have characteristics such as gender, number etc. When all such required details are grouped together with the noun, the resulting structure is known as noun phrase (NP). The basic structure of Urdu noun phrase is, “(M) H (M)” according to (Butt M., 1995), where (M) is a modifier and (H) is the head of a NP. Head is the word which is compulsory and modifiers can or cannot be there. In Urdu modifiers are of two types pre-modifiers i.e modifiers that come before the head for instance (کالی بلی kali: bli: “black cat”), and post-modifiers which come after the head for instance (تم سب tm sb “you all”). In GF resource library we represent NP as a record

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

where

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

Thus NP is a record with two fields, ’s’ and ’a’. ‘s’ is an inflection table and stores different forms of a noun.

The Urdu NP has a system of syntactic cases which is partly different from the morphological cases of the category noun (N). The case markers that follow nouns in the form of post-positions cannot be handled at lexical level through morphological suffixes and are thus handled at syntactic level (Butt et el., 2002). Here we create different forms of a noun phrase to handle case markers for Urdu nouns. Here is a short description of the different cases of NP :

- NPC Case: this is used to retain the original case of Noun
- NPErg: Ergative case with case marker 'ne: نے'
- NPAbl: Ablative with case marker 'se: سے'
- NPIns: Instrumental case with case marker 'se: سے'
- NPLoc1: Locative case with case marker 'mi: ɳ میں'
- NPLoc2: Locative case with case marker 'pr پر'
- NPDat: Dative case with case marker 'kʊ کو'
- NPAcc: Accusative case with case marker 'kʊ کو'

And 'a' (Agr in the code sample given in previous column) is the agreement feature of the the noun that is used for selecting the appropriate form of other categories that agree with nouns.

A noun is converted to an intermediate category common noun (CN; also known as N-Bar) which is then converted to NP category. CN deals with nouns and their modifiers. As an example consider adjectival modification:

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

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

The linearization of AdjCN gives us common nouns such as (ٹھنڈا پانی ʈ^{h}n ɖa pani: "cold water") where a CN (پانی pani: "water") is modified by an AP (ٹھنڈا , ʈ^{h}n ɖa "cold").

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

A CN can be converted to a NP using different functions: common nouns with determiners; proper names; pronouns; and bare nouns as mass terms:

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

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

4.2 Verb Phrases (VP)

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

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

where

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

and

```
  VPPTense = VPPres |VPPast |VPFutr;
 HLevel = Tu |Tum |Ap |Neutr
```

In GF representation a VP is a record with different fields. The most important field is ‘s’ which is an inflectional table and stores different forms of Verb.

At VP level we define Urdu tenses by using a simplified tense system, which has only three tenses, named VPPres, VPPast, VPFutr. In case of VPTense for every possible combination of VPPTense and agreement (gender, number, person) a tuple of two string values {fin, inf : Str} is created. ‘fin’ stores the coupla (auxiliary verb) , and ‘inf’ stores corresponding form of verb. VPStem is a special tense which stores the root form of verb. This form is used to create the full set of Urdu tenses at clause level (tenses in which the root form of verb is used, i.e. perfective and progressive tenses). Handling tenses at clause level rather than at verb phrase level simplifies the VP and results in a more efficient grammar.

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

Other forms for verb phrases include request form (VPReq), imperative form (VPImp). There are four levels of requests in Urdu. Three of them correspond to (tʋ تو, tm تم , a:p آپ) honor levels and the fourth is neutral with respect to honorific levels. .

The Urdu VP is a complex structure that has different parts: the main part is a verb and then there are other auxiliaries attached to verb. For example an adverb can be attached to a verb as a modifier. We have a special field ‘ad’ in our VP representation. It is a simple string that can be attached with the verb to build a modified verb. In Urdu the complement of a verb precedes the actual verb e.g (وہ دوڑنا چاہتی ہے ʋo dʊɽna tʃahti: he: “she want to run”), here (چاہنا tʃahna “want”) is complement of verb (دوڑنا dʊɽna “run”), except in the case where, a sentence or a question is the complement of the verb. In that case complement of the verb comes at the very end of clause e.g (ʋo khta he: kh ʋo dʊɽti: he: وہ کہتا ہے کہ وہ دوڑتی ہے “he says that she runs”). We have two different fields named ‘compl’ and ‘embCompl’ in the VP to deal with these different situations.

‘vType’ field is used to store information about type of a verb. In Urdu a verb can be transitive, intransitive or double-transitive (Schmidt R. L., 1999). This information is important when dealing with ergativity in verb agreement. The information about the object of the verb is stored in ‘obj’ field. All this information that a VP carries is used when a VP is used in the construction of a clause.

A distinguishing feature of Urdu verb agreement is ‘ergativity’. Urdu is one of those languages that shows split ergativity at verb level. Final verb agreement is with direct subjective except in the transitive perfective tense. In transitive perfective tense verb agreement is with direct object. In this case the subject takes the ergative construction (subject with addition of ergative case marker (ne: نے).

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

لڑکا کتاب خرید چکا ہے
lɽka $_{\text{Direct}}$ ktab $_{\text{Direct}}$ xri:d $_{\text{Root}}$ tʃka $_{\text{aux\_verb}}$ he:
The boy has bought a book

The second way to make the same clause is

لڑکے نے کتاب خریدی ہے
lɽke: ne: $_{\text{Erg}}$ ktab $_{\text{Direct\_Fem}}$ xri:di: $_{\text{Direct\_Fem}}$ he:
The boy has bought a book

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

In the concrete syntax we ensure this ergative behavior through the following code segment in GF. However the code given here is just a segment of the code that is relevant.

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

e.g in case of simple past tense if verb is transitive then ergative case of noun is used and agreement is with object of verb. In all other cases direct case of noun is used and agreement is with subject of verb.

A VP is constructed in different ways; the simplest is

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

where V is the morphological category and VP is the syntactic category. There are other ways to make a VP from other categories, or combinations of categories. For example

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

An adverb can be attached to a VP to make an adverbial modified VP. For example (i:haŋ یہاں سونا)

4.3 Adjective Phrases (AP)

Adjectives (A) are converted into the much richer category adjectival phrases (AP) at syntax level. The simplest function to convert is

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

Its linearization is very simple, since in our case AP is similar to A e.g.

```
fun PositA a = a ;
```

There are other ways of making AP for example

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

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

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

4.4 Clauses

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

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

The subject-verb agreement is insured through agreement feature of NP which is passed to verb as inherent feature. A clause is of following type

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

Here VPHTense represents different tenses in Urdu. Even though current abstract level of common API does not cover all tenses of Urdu, we cover them at clause level and can be accessed through language specific module. So, VPHTense is of following type

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

Polarity is used to make positive and negative sentences; Order is used to make simple and interrogative sentences. These parameters are of following forms

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

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

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

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

4.5 Question Clauses and Question Sentences

Common API provides different ways to create question clauses. The simplest way is to create from simple clause

fun QuestCl : Cl -> QCl ;

In Urdu simple interrogative sentences are created by just adding "ki:a کیا" at the start of a direct clause that already have been created at clause level. Hence, the linearization of above function simply selects appropriate form of clause and adds "ki:a کیا" at the start. However this clause still has variable tense and polarity which is fixed at sentence level e.g

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

Other forms of question clauses include clauses made with interrogative pronouns (IP), interrogative adverbs (IAdv), and interrogative determiners (IDet), categories. Some of the functions for creating question clauses are

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

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

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

5. Example

As an example consider the translation of following sentence from English to Urdu, to see how our proposed system works at different levels.

He drinks hot milk.

Figure 1 shows the parse tree for this sentence. As a resource grammar developer our goal is to provide correct concrete level linearization of this tree for Urdu.

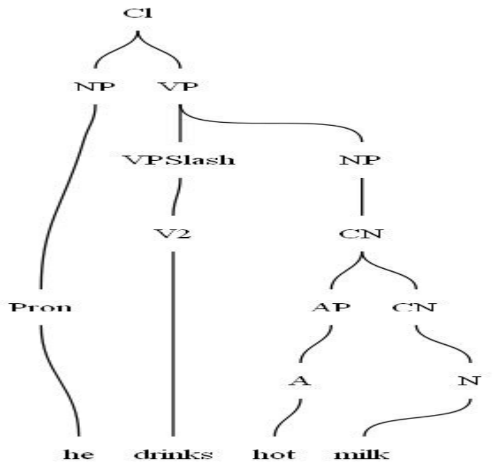

Figure 1. Parse tree of an example sentence

The nodes in this tree represent different categories and its branching shows how a particular category is built from other categories and/or leaves (words from lexicon). In GF notation these are the syntactic rules which are declared at abstract level. For example category CN can be built from an AP (adjectival phrase) and a CN. So in GF representation it has following type signature.

fun AdjCN : AP -> CN -> CN ;

A correct implementation of this rule in Urdu concrete syntax ensures correct formation of a common noun (گرم دودھ grm dʊdʱ "hot milk") from a CN (دودھ dʊdʱ "milk") modified by an Adjective (گرم , grm "hot").

A NP is constructed from this CN by one of the NP construction rules (see section 4.1 for details). A VPSlash (object missing VP) is build from a two place verb (پیتا pi:ta “drinks”). This VPSlash is then converted to VP through function

fun ComplSlash : VPSlash -> NP -> VP ;

Resulting VP and NP are grouped together to make a VP (گرم دودھ پیتا ہے ʈgrm dʊdʱ pi:ta he: “drinks hot milk”). Finally clause (گرم دودھ پیتا ہے وہ ʊh grm dʊdʱ pi:ta he: “he drinks hot milk”) is build from NP (وہ ʊh “he”) which is build from pronoun (وہ ʊh “he”) and VP (گرم دودھ پیتا ہے grm dʊdʱ pi:ta he: “drinks hot milk”). Language dependent concrete syntax assures that correct forms of words are selected from lexicon and word order is according to rules of that specific language. While, morphology makes sure that correct forms of words are built during lexicon development.

6. An application: Attempto

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

7. Related Work

A suite of Urdu resources were reported in (Humayoun et el., 2006) including a fairly complete open-source Urdu morphology and a small fragment of syntax in GF. In this sense, it is a predecessor of Urdu resource grammar, implemented in a different but related formalism.

Like the GF resource library, Pargram project (Butt et el., 2007) aims at building a set of parallel grammars including Urdu. The grammars in Pargram are connected with each other by transfer functions, rather than a common representation. Further, the Urdu grammar is still one of the least implemented grammars in Pargram at the moment. This project is based on the theoretical framework of lexical functional grammar (LFG).

Other than Pargram, most work is based on LFG and translation is unidirectional i.e. from English to Urdu only. For instance, English to Urdu MT System is developed under the Urdu Localization Project (Hussain, 2004), (Sarfraz and Naseem, 2007) and (Khalid et el., 2009).

Similarly, (Zafer and Masood, 2009) reports another English-Urdu MT system developed with example based approach. On the other hand, (Sinha and Mahesh, 2009) presents a strategy for deriving Urdu sentences from English-Hindi MT system. However, it seems to be a partial solution to the problem.

8. Future Work

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

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

9. Conclusion

The resource grammar we develop consists of 44 categories and 190 functions[3] which cover a fair enough part of language and is enough for

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

building domain specific application grammars including multilingual dialogue systems, controlled language translation, software localization etc. Since a common API for multiple languages is provided, this grammar is useful in applications where we need to parse and translate the text from one to many other languages.

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

10. References

Angelov K. and Ranta A. 2010. *Implementing controlled Languages in GF.* Controlled Natural Language (CNL) 2009, LNCS/LNAI Vol. 5972 (To appear)

Attempto 2008. Project Homepage. attempto.ifi.uzh.ch/site/

Butt M., 1995. *The Structures of Complex Predicate in Hindi* Stanford: CSLI Publications

Butt M., Dyvik H., King T. H., Masuichi H., and Rohrer C. 2002. *The Parallel Grammar Project.* In Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation. pp. 1-7.

Butt, M. and King, T. H. 2007. *Urdu in a Parallel Grammar Development Environment'*. In T. Takenobu and C.-R. Huang (eds.) *Language Resources and Evaluation*: Special Issue on Asian Language Processing: State of the Art Resources and Processing 41:191-207.

Forsberg M., and Ranta A., 2004. *Functional Morphology.* Proceedings of the Ninth ACM SIGPLAN International Conference of Functional Programming, Snowbird, Utah.

Humayoun M., Hammarström H., and Ranta A. *Urdu Morphology, Orthography and Lexicon Extraction. CAASL-2:* The Second Workshop on Computational Approaches to Arabic Script-based Languages, July 21-22, 2007, LSA 2007 Linguistic Institute, Stanford University. 2007

Hussain, S. 2004. *Urdu Localization Project. COLING*:WORKSHOP ON Computational Approaches to Arabic Script-based Languages, Geneva. pp. 80-81

Khalid, U., Karamat, N., Iqbal, S. and Hussain, S. 2009. *Semi-Automatic Lexical Functional Grammar Development.* Proceedings of the Conference on Language & Technology 2009.

Masica C., 1991. *The Indo-Aryan Languages, Cambridge,* Cambridge University Press, ISBN 9780521299442.

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

Ranta A. *The GF Resource Grammar Library A systematic presentation of the library from the linguistic point of view.* to appear in the on-line journal Linguistics in Language Technology, 2009a.

Ranta A. *Grammars as Software Libraries. From Semantics to Computer Science,* Cambridge University Press, Cambridge, pp. 281-308, 2009b.

Rizvi, S. M. J. 2007. *Development of Algorithms and Computational Grammar of Urdu.* Department of Computer & Information Sciences/ Pakistan Institute of Engineering and Applied Sciences Nilore Islamabad. Pakistan.

Sarfraz H. and Naseem T., 2007. *Sentence Segmentation and Segment Re-Ordering for English to Urdu Machine Translation.* In Proceedings of the Conference on Language and Technology, August 07-11, 2007, University of Peshawar, Pakistan.

Schmidt R. L., 1999. *Urdu an Essential Grammar,*Routledge Grammars.

Sinha R., and Mahesh K., 2009. *Developing English-Urdu Machine Translation Via Hind.,* Third Workshop on Computational Approaches to Arabic Script-based Languages (CAASL3) in conjunction with The twelfth Machine Translation Summit. Ottawa, Ontario, Canada.

Zafar M. and Masood A., 2009. *Interactive English to Urdu Machine Translation using Example-Based Approach.* International Journal on Computer Science and Engineering Vol.1(3), 2009, pp 275-282.