Urdu Morphology and Beyond: Why Grammars Should not Live without Finite-State Methods

(Draft)

Tina Bögel, Miriam Butt, and Tracy Holloway King

1 Introduction

> The first thing I ever said to Lauri Karttunen was: "Nice to meet you. I fell in love with your FSM book". He blushed. And smiled.

This paper examines the role that Finite State Transducers (FST) play in the Urdu ParGram grammar.[1] Like most of the ParGram grammars, the Urdu grammar uses FSTs for **tokenization** and for **morphological analysis**. In addition to these basic components, the Urdu grammar is special among the ParGram grammars in that it also integrates a **transliteration** FST. This transliteration component allows the Urdu grammar to function as the core engine for the parsing and generation of both Urdu and Hindi. Urdu and Hindi are structurally similar enough that they are often treated as versions of one language. However, Urdu is written in a version of the Arabic script and Hindi is written in Devanagari. The transliteration component takes input from both scripts and converts it into an underlying Latin transliteration that can be used for both languages.

Urdu/Hindi has relatively complex morphology. Our implementation of the Urdu/Hindi morphological component uses all the capabilities of finite-state morphology (FSM) introduced by Beesley and Karttunen (2003), including **phonological rules** and **reduplication**. These interact in a complex manner with the FST transliteration component. We also use the morphological component for the dynamic formation of **morphological complex predicates**, i.e., to model the Urdu/Hindi morphological causative. Complex predication in the Urdu grammar (Butt et al., 2003, 2009) is implemented in the syntax via the restriction operator (Kaplan and Wedekind, 1993) and phrase structure rules and in the morphological component via sublexical rules. The implementation uses standard capabilities provided by FSM and its interface to the syntax in the grammar development platform XLE (Crouch et al., 2016, Kaplan et al., 2004). However, it raises questions with respect to the interaction of lexical rules with the restriction

[1]We would like to thank Tafseer Ahmed, Ghulam Raza, and an anonymous reviewer for their help. Parts of this research was funded by the Urdu ParGram project funded by the Deutsche Forschungsgemeinschaft. Research for this paper by the third author was done prior to her joining Amazon.

operator. In particular, it raises issues of how morphology and syntax interface with respect to argument structure operations such as passivization and causativization.

We address these topics as part of the paper. The system architecture is presented in section 2, the core tokenization in section 3, the script transliterator in section 4 and the morphological component in section 5. Section 6 describes how these are specified in the XLE implementation. The interaction between morphology and syntax with respect to complex predicate formation is the topic of section 7. Section 8 concludes that an integration of FSTs into a computational grammar taps into a powerful yet intuitive and user-friendly technology that is compatible with theoretical proposals concerning the role of morphology in a modular architecture of grammar (Karttunen, 2003, Dalrymple, 2015). It is thus a technology that no computational grammar should have to do without.

The major conclusion of our paper is that we, as a field, are indebted to Lauri Karttunen for playing a crucial role in developing this theory-driven technology. Without his combination of theoretical and implementational genius, this technology would not have been taken as far and would not have become as available to the community in its intuitive and accessible manner.[2]

2 System Architecture

The Urdu grammar (Butt and King, 2002) is part of the Parallel Grammar (ParGram) project (Butt et al., 1999, 2002). The ParGram project originally focused on three closely related European languages: English, French, and German. Once grammars for these languages were established, more languages were added, including Urdu. Grammars have been developed for Arabic, Georgian, Hungarian, Indonesian, Irish, Japanese, Malagasy, Norwegian, Polish, Tigrinya, Turkish, Welsh and Wolof. The ParGram project uses the XLE parser and grammar development platform (Maxwell III and Kaplan, 1993, Crouch et al., 2016). The grammars use the Lexical-Functional Grammar (LFG) formalism which produces c(onstituent)-structures (trees) and f(unctional)-structures (attribute-value matrices or AVMs) as syntactic analyses. A parallel treebank across several of the ParGram languages has been developed (Sulger et al., 2013) and is accessible via the INESS treebanking technology (Rosén et al., 2009, 2012b,a).

LFG assumes a version of Chomsky's Universal Grammar hypothesis that all languages are governed by similar underlying structures. Within LFG, f-structures encode a language universal level of analysis, reflecting

[2]We are personally immeasurably indebted to Lauri Karttunen for his long-term involvement in the ParGram project, his avid interest in different applications for FSTs, and his boundless encouragement for our work.

cross-linguistic parallelism. The ParGram project aims to test the LFG formalism for its universality and coverage limitations, determining how far parallelism can be maintained across languages. Where possible, the analyses produced for similar constructions in each language are parallel. The standardization of the analyses has the computational advantage that the grammars can be used in similar applications and this standardization can simplify cross-language applications (Frank, 1999).

Most of the ParGram grammars use finite-state tokenizers and morphologies (Beesley and Karttunen, 2003). None of them uses a full-form lexicon. Having access to a morphological analyzer is crucial for large-scale, robust coverage for morphologically complex languages.

In this paper, we focus on the roles FSTs play in the Urdu ParGram grammar. There are three types of FSTs in the Urdu grammar: an Urdu↔Latin transliterator, a tokenizer, and a morphology. Within the ParGram context, the transliteration step is unique: it was introduced to allow the grammar to parse both Urdu, written in Arabic script, and Hindi, written in Devanagari,[3] using Latin alphabet forms as a lingua franca.

$$\begin{tikzcd}
\text{Tokenizer + Transliterator (FST)} \arrow[d] \\
\text{Morphology (FST)} \arrow[d] \\
\text{LFG ParGram Grammar}
\end{tikzcd}$$

Figure 1: Configuation of the FSTs in the Urdu Grammar (Parsing)

The following sections discuss the components in Figure 1 from the perspective of parsing. LFG grammars are reversible and hence are also used for generation. That is, an f-structure can be given as input to a grammar and the appropriate strings generated. We focus mainly on parsing within this papers. See Maxwell III (2006), Kaplan and Wedekind (2000), Wedekind and Kaplan (2012), Zarrieß et al. (2011) and Crouch et al. (2016) on LFG and generation.

3 Tokenizer

The first step in the natural language processing (NLP) of text is that of tokenization, i.e., of breaking a text into words or tokens. Languages differ in the punctuation systems they use as well as in the conventions used to indicate word boundaries. Therefore, many of the ParGram grammars have developed specialized tokenizers. The Urdu grammar integrates a cascade of

[3]We do not discuss Devanagari here, focusing on the Urdu transliteration which is a significantly more complex task.

FSTs, including a transliterator, a morphological analyzer and a component for the identification of multi-word expressions (MWE). The special needs of Urdu are taken care of by the combination of these components and the default XLE tokenizer that is included as part of the XLE grammar development platform (Crouch et al., 2016). This default tokenizer is based on the tokenizer described in chapter 9 of the FSM book (Beesley and Karttunen, 2003) and has been elegantly augmented by Ron Kaplan (Kaplan, 2005, Forst and Kaplan, 2006).

As a standard FST, the tokenizer is bidirectional. In the Urdu grammar, the tokenizer is used for parsing and generation. In order to parse and generate Urdu script, it is combined with the transliterator described in the next section.

4 Transliterator

One of the characteristics of the Urdu ParGram grammar is its aim to cover both Urdu and Hindi. The architectural design involves a core grammar written in Latin script flanked by two FSTs, one for Urdu script and one for Devanagari, which makes it possible to switch from one language to the other while using a single underlying grammar. This section describes the Urdu FS transliterator[4] and how it deals with Urdu-specific orthographic issues by integrating restrictional components to minimize the generation of words not found in Urdu. The transliterator is based on a non-probabilistic FST compiled with the **lexc** language (Lexicon Compiler), which is designed to build finite state networks and analyzers implemented with the Xerox finite state technology (XFST) (Beesley and Karttunen, 2003). The resulting network is compatible with one written with, e.g., regular expressions, but is human readable. The underlying transliteration scheme was developed by Malik et al. (2010), following Glassman (1986).

4.1 Urdu script issues

Urdu is written in a version of the Persian alphabet (itself derived from the Arabic alphabet). The direction of the script is from right to left and the shapes of most characters are context sensitive, i.e., depending on its position within the word a character assumes a certain form.

To transliterate from Urdu to Latin script and vice versa is not a one-to-one transliteration. The main reason for this are the diacritics and semivowels. Table 4.1 represents the four most frequent diacritics (of a total of 15; Malik (2006, 13)) in combination with the letter ب 'b' and the sounds represented by the semi-vowels

[4]This section is based extensively on Bögel (2012), written by the first author.

ي ‘yeh’ and و ‘wao’, respectively.

ب + diacritic	Name	Latin transliteration
بَ	Zabar	ba
بِ	Zer	bi
بُ	Pesh	bu
بّ	Tashdid	bb
Semivowels		
و	Wao	v, O, U, o
ي	Yeh	y, E, I, e

Table 1: The four most frequently used Urdu diacritics and the two semivowels

The vowels signified by the diacritics are always short. Long vowels are indicated by capital letters (e.g., ‘E’); short vowels are in lower case (e.g., ‘e’). The Tashdid indicates gemination.

A difficulty for Urdu NLP is that the diacritics are generally not used in Urdu text. Coupled with the range of interpretative possibilities for the semivowels, this results in the (over-)generation of words which are not part of the Urdu language. Consider Figure 2, where the word کتا kuttA ‘dog’ is to be transliterated. The word contains two vowels and a geminated consonant. Without diacritics, the written word consists of just three letters: *k*, *t* and *A*. If the transliteration system hypothesizes all possible short vowels and geminated consonants, there are six possibilities.

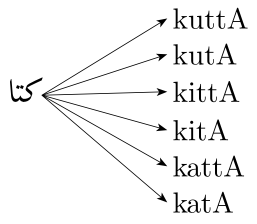


Figure 2: Overgeneration for the sequence کتا kuttA ‘dog’

Phonotactic constraints usually reduce the number of forms. For example, Urdu does not generally allow consonant clusters, with a schwa added to loan words like *school* or *station* to make them conform to Urdu phonotactics (*səkul* or *əsteʃon*). Strings such as `ktA` therefore do not form part of the search space that needs to be considered for words as in Figure 2. Furthermore, final short vowels should not be hypothesized as they are always indicated in the script (*`ktAi`). These and other phonotactic constraints were implemented in transliterator to reduce the number of possible outcomes.

However, even with phonotactic constraints, there is still usually more than one logically possible option for transliteration, often several, as in Figure 2. One way to reduce the number of possibilities are lists of Urdu words that the system can match against. However, static word lists do not allow for alternate spellings and exclude 'mixed' words in which some of the diacritics are written, but not all of them. Furthermore, a word list is never complete, as languages constantly coin or importing new words. English has been a major source of new lexical items in Urdu, due to intensive language contact over several centuries. Urdu texts thus often include English words in Latin script.

As a result of these considerations, the Urdu-Latin transliterator was designed to include: (a) a word list against which known words are matched; (b) a default transliteration component that allows the transliteration of unknown words in the Urdu script. Furthermore, the system includes a default Latin 'transliterator' to allow for words in the Latin script to be parsed as well. The next section describes these components in detail.

4.2 Basic components

The Urdu transliterator integrates three layers of restrictions that constrain the transliteration possibilities for a word in Urdu script.

First, a word list is incorporated. The FS transliterator works with a Latin script list of Urdu words derived from two resources: (a) a word list that was compiled in the process of creating a dictionary and was shared with us by our colleagues from CRULP;[5] (b) a word list compiled from the Urdu finite state morphology (Bögel et al., 2007, 2009) (section 5). The second word list increases automatically whenever an item is added to the Urdu finite state morphology.

Second, based on the phonotactics of the language, regular expression filters reduce the possibilities proposed by the transliterator. Consider the filter in (1).

(1) [~[y A [a |i |u]]]

In Urdu a combination of [*y A short vowel*] is not allowed, as indicated by the negation (complement) symbol ~. A filter like (1) disallows all generations that match the sequence.

Third, the FST must be able to deal with unknown items. In the UrduGram FST architecture, items that do not match the word list are first

[5]The Center for Research in Urdu Langauge Processing (CRULP) was located at the National University of Computer and Emerging Sciences in Lahore, Pakistan. It has been superseded by the Center for Language Engineering (CLE) at the University of Engineering and Technology in Lahore, Pakistan (http://www.cle.org.pk/).

passed through the phonotactic constraints filter before being fed into the default transliterator, thus allowing a maximum number of transliterations while preventing extensive overgeneration.

These three components are implemented with the finite state techniques proposed in Beesley and Karttunen (2003). To compose these within a single network, while maintainging efficiency, several layers of the transliterator reused during the transliteration process.

4.3 The Overall Transliterator Architecture

Figure 3 illustrates typical instances of transliteration with respect to two sequences: کتاب *kitAb* 'book' and کت. The latter transliterates to an unknown word *kt*, potentially corresponding to the surface forms *kut, kat* or *kit.* Recall that neither consonant clusters nor final short vowels should be hypothesized by the system (cf. section 4.2).

Step 1: Transliteration part 1 Step 1 of the FST contains a character mapping from Urdu to Latin (left column) and an identity mapping component for both Urdu and Latin (right column). Even though phonotactic filters are applied during the Urdu-Latin mapping, the output of the basic transliterator shows (part of) the overgeneration caused by the underspecified nature of the script, e.g., for the word
کتاب, a total of 24 transliterations are possible.

The network also contains a default Urdu and a default Latin component (right column) where the respective characters are matched against themselves (e.g. k:k, r:r). That is, an Urdu script word will not only be transliterated into the corresponding Latin script, but will also be 'transliterated' into itself plus an identificational tag. The Urdu script default one-to-one mappings are marked with a special identification tag (`[+Uscript]`) for further processing.

Step 2: Word list matching and tag deletion In step 2, the output is matched against the Latin word list (s.a.). When there is a match (as it is the case for *kitAb*), the respective word is tagged `[+match]`. Following the matching, a filter is applied which erases all output forms that contain no tags, neither `[+match]` nor `[Uscript+]`. Consequently, there are two

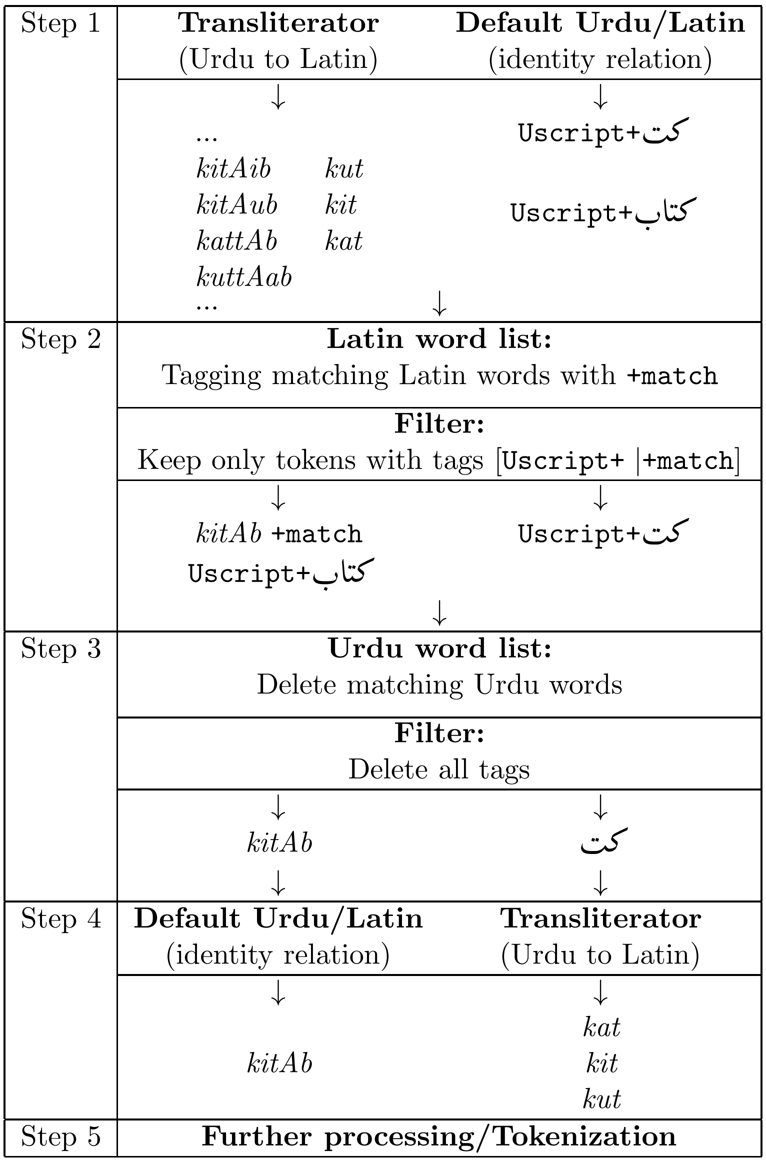

Step 1
Transliterator (Urdu to Latin)
Default Urdu/Latin (identity relation)
...
kitAib kut
kitAub kit
kattAb kat
kuttAab
...
Uscript+کت
Uscript+کتاب
Step 2
Latin word list:
Tagging matching Latin words with +match
Filter:
Keep only tokens with tags [Uscript+ |+match]
kitAb +match
Uscript+کتاب
Uscript+کت
Step 3
Urdu word list:
Delete matching Urdu words
Filter:
Delete all tags
kitAb
کت
Step 4
Default Urdu/Latin (identity relation)
Transliterator (Urdu to Latin)
kitAb
kat
kit
kut
Step 5
Further processing/Tokenization

Figure 3: Transliteration of کت and کتاب

choices left for کتاب *ktAb* (one transliterated 'matched' form and one default Urdu form) while کت *ktb* is left with only the default Urdu form.

Step 3: Distinguishing unknown from overgenerated entities The Urdu word list applied in step 3 is a transliteration of the original Latin word list, which was transliterated via the present FS transliterator system. That is, the Urdu word list is a mirror image of the Latin word list. During step 3, the Urdu script words are matched against the Urdu word list, *deleting* all the words that *find* a match. By definition, all of the words that found a match in the Latin word list in step 2 will also find a match in the Urdu word list in step 3, while all unknown entities will fail to match. As a result, any Urdu script version of an already correctly transliterated word is deleted, while the Urdu script of an unknown entity is kept for further processing. Finally, the tags of the remaining entities are deleted, which leaves the correct transliteration of the known word *kitAb* and the unknown Urdu script word کت.

Step 4: Transliteration part 2 The remaining words are again sent into the FST of step 1. The Latin transliteration *kitAb* passes through the default Latin transliterator, effectively matching *kitAb* against *kitAb*. The unknown Urdu script word is transliterated into all three possible forms *kit, kut, kat.*

Step 5: Final adjustments So far, the transliterator only applies to single words. To allow the transliterator to process entire texts, the FST network is composed with a standard tokenizer (section 3). Effectively, the tokenizer first breaks the text into words and then the transliterator applies to each word. The resulting composed FST is used within the Urdu ParGram MCONFIG, which specifies the tokenizers and morphologies used for parsing and generation (see Section 6).

4.4 Transliteration Summary

The Finite State Transducer for Urdu $\leftrightarrow$ Latin transliteration described above is a constrained, yet flexible, efficient, and robust system. The price of ensuring dynamic flexibility in the transliteration from Urdu to Latin is that the system generates non-existent words due to the underspecified nature of the Urdu script. This overgeneration is reduced by applying layers of restrictions based on Urdu phonotactics and the incorporation of word lists. The transliterator thus distinguishes between items unknown to the word list (*kat, kut, kit*) and non-existent items like *kittAub*, thus restricting the possible combinations while always producing at least one transliteration.

The nature of an FST is that parsing input equals generation output. For the transliterator this means that if diacritics are optionally parsed, then they will be optionally generated. To avoid this, two transliterators were created for generation: one which obligatorily generates Urdu script with diacritics and one which generates text without any diacritics; mixed cases are not allowed. Both generators use the architecture described above and only require minimal adjustments. The transliterator can also be used to convert text without diacritics to text that contains diacritics by parsing a text without diacritics and then generating one with diacritics. Texts with partial diacritics can be transliterated and then generated either as fully without or fully with diacritics.

5 Morphological Analysis

Urdu ParGram grammar provides a basis for processing both Urdu and Devanagari scripts by means of transliterators to Latin script. Thus, the input to the Urdu morphological analyzer must be in Latin script. The morphological analyzer is similar to those used in other ParGram grammars and was custom-built for use in the Urdu grammar following the guidelines in Beesley and Karttunen (2003).[6] Although the morphology was built for the Urdu grammar, it was designed to function as a stand-alone module. For example, it can be used as a lemmatizer, to provide potential parts of speech, and as a morphological analyzer for other grammars.

The Urdu morphological analyzer associates surface forms of words with a canonical form (a lemma) and a series of morphological tags that provide grammatical information about that form. An example for English is shown in (2) and for Urdu in (3).

(2)	pushes:	push +Verb +Pres +3sg
		push +Noun +Pl

(3) bOlA bOl +Verb +Perf +Masc +Sg

(2) states the English *pushes* can either be the third singular form of the verb *push* or the plural of the noun *push*. (3) states that the Urdu *bOlA* is the perfect masculine singular form of the verb *bOl* 'speak'.

In developing the Urdu morphological analyzer, we found that the finite-state tools and solutions to problems like the application of phonological rules or reduplication outlined in Beesley and Karttunen (2003) met the challenges posed by the Urdu morphological system.[7]

[6]Not all ParGram grammars use custom-built morphologies: some ParGram grammars (e.g. English, German) incorporate existing morphological analyzers (Butt et al., 1999).

[7]This includes derivational morphology and compounding.

Part of Speech	**Number of lemmata**
Nouns	210
Verbs	100
Adjectives	124
Adverbs	71

Table 2: Lemma coverage of the Urdu/Hindi morphology

5.1 Morphological Coverage

The morphology covers all the inflected forms for nouns, verbs, adjectives, and adverbs, as well as closed-class words such as postpositions and conjunctions. Table 2 shows how many lemmata of the major verb classes are covered by the morphology. The focus in building the morphological analyzer was to cover all of the types of verbal paradigms. This work is complete and now new lemmata can be added very simply into the existing word classes.

Beesley and Karttunen's (2003 definition of finite-state morphology allows for non-concatenative morphology, the integration of phonological rules, long-distance dependencies and a treatment of reduplication. The Urdu morphological analyzer depends on all of these, particularly phonological rules and reduplication (Bögel et al., 2007). It also makes heavy use of flags for long-distance dependencies.

The number of simple verbs in Urdu is between 500–800 (Humayoun, 2006). The majority of verbal predication is achieved via complex predicates, which can be composed of nouns, verbs, adjectives or adpositions in combination with a light verb (Butt, 1995, Mohanan, 1994). A significant part of the Urdu grammar development effort concerned the treatment of complex predicates (Hautli et al., 2012, Ahmed et al., 2012, Butt et al., 2012, Hautli-Janisz, 2014, Sulger and Vaidya, 2014, Hautli-Janisz et al., 2015). Morphological issues around predicate formation arose with respect to morphological causative complex predicates (section 7).

In addition to the lemmata specified in the morphology, the morphology contains a guesser. In many of the ParGram grammars, this guesser is a separate FST which processes words which receive no analysis from the core finite state morphology. In the Urdu grammar, there is a single finite-state morphology which contains both the known words and a guesser to handle forms unknown by the morphology. The guesser was designed primarily for unknown names, but is also able to guess verbs and adjectives. The guesser was designed according to the specifications in chapter 9 of Beesley and Karttunen (2003).

The Urdu morphological analyzer represents a realizational morphology (Karttunen, 2003), which is formally equivalent to Paradigm Function

Morphology, an inferential realizational theory of inflectional morphology (Stewart and Stump, 2006). As such, several pieces of morphology may correspond to one feature, or one piece of abstract morphological information may be "realized" by several different overt markers. Our design decisions are consonant with this approach.

Several issues arose around the design of the morphological tag set. Recall from (2) and (3) that the morphological analyzer associates a surface form with a lemma and a set of tags. These tags are a combination of part-of-speech information such as +Verb, +Noun or +Country[8] and tags representing information such as +Fem, +Sg and +3P for feminine, singular and third person. Most of the tags follow straightforwardly from the lingusitically motivated morphological analysis of Urdu. However, we faced several design decisions.

First, it is possible to combine different types of information into one tag, as in the +3sg tag in the English example in (2). However, we separated out all of pieces of morphological information since that led to an interface that is independent of the Urdu ParGram grammar.

Second, althoughwe generally designed the morphological analyzer to be as explicit as possible, in some cases explicit analyses lead to unncessary ambiguity. Consider the future forms. The future is formed via the verb plus subjunctive morphology which also provides information about number and person. This is followed by the future marker *g*, which is inflected for number and gender morphology (*A/I/E*). An example is shown in (4). In Hindi the verb and the future marking are written together, while in Urdu they are written as separate words. As such, examples as in (4) present an issue for tokenization (section 3) and morphological analysis. We normalize the different writing practices in our transliterator and treat the verb plus its future marking as one lexical item.

(4) mArUNgI ⇔
mAr+Verb+Subjunct+1P+Sg+Fut+Fem

From the perspective of the syntax, marking the form as both subjunctive and future is redundant: every future form also carries subjunctive meaning. Experience gathered with respect to the German grammar (Butt et al., 1999) showed that it is better to eliminate tags of this kind from the morphology, since dealing with them complicates the morphology-syntax interface. For example, Urdu also uses the subjunctive morphology as a real subjunctive, see (5).

(5) mArUN ⇔
mAr+Verb+Subjunct+1P+Sg

[8]The latter are particularly useful for named entity recognition applications, hence we included them as part of the morphological analysis.

Given that these subjunctive uses exist in parallel with the future use in (4), the interpretation of the +Subjunct tag by the morphology-syntax interface would need to differ depending on whether it is found in conjunction with future morphology or not, thus complicating the interface. We therefore eliminated the +Subjunct tag from the morphological analysis of future forms, allowing for a cleaner morphology-syntax-semantics interface.

This same problem is found with respect to infinitives as in *dEkHnA* ‘to look/looking’,[9] which are used as verbal nouns. The morphology could potentially provide both the analyses in (6).

(6) dEkHnA ⇔
  a. dEkH+Verb+Inf+Masc+Sg
  b. dEkH+Noun+Deverb+Masc+Sg

However, this would result in unnecessary ambiguity since all infinitives are also deverbal nouns. Instead, this option has been encoded as part of the morphology-syntax interface (next section). The entry in (7) shows how the grammar interprets the abstract tag +Inf. The tag is associated with a functional annotation that states that words with this tag can optionally (denoted by the round brackets) be used as a noun whose type is deverbal.

(7) +Inf   (($\uparrow$ NTYPE) = deverbal).

This solution pushes the ambiguity from the morphology into the syntax, where syntax can resolve the ambiguity via other information from the clausal context.

5.2 Morphology-Syntax Interface

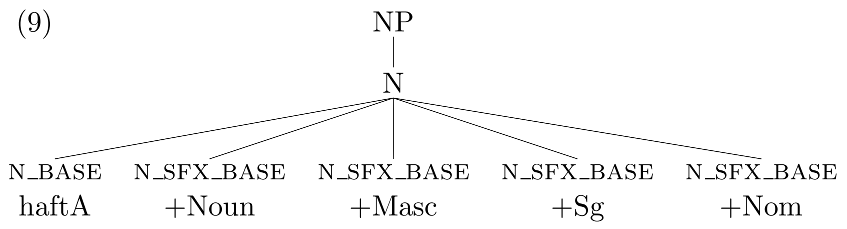
In this section, we show how the output of the Urdu FST morphology is integrated into the Urdu LFG grammar. The morphology-syntax interface is as described in Kaplan et al. (2004). The intuition behind the interface is that the lemma and tags can be treated on a par with lexical items. For example, in the lexicon, a tag +Past can be associated with the information that the value of tense is past, as in (8).

(8) +Past ($\uparrow$ TNS-ASP TENSE)=past

The “lexical items” coming out of the morphology, namely the lemma and the tags, are assembed via sublexical rules into a word. LFG grammars assume the notion of Lexical Integrity, which means that the terminal leaves are syntactically independent items (fully formed words or clitics). With the integration of a finite-state morphology, these terminal nodes of fully formed

[9]Capital H in the transliteration indicates aspiration of the preceding consonant.

lexical items are replaced by the output of the morphology. The grammar uses the output of the morphology as the leaves of the c(onstituent)-structure tree. For example, the noun *haftA* 'week' forms a c-structure tree as in (9).

(9)


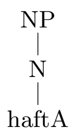
The standard XLE LFG context-free rule formalism is used to assemble the lemmata and tags into a sublexical tree as in (9). A sample context free c-structure rewrite rule that produces the tree in (9) is shown in (10). The rule states that an N is composed of an N_BASE and at least one N_SFX_BASE as indicated by the Kleene +.[10]

(10) N $\longrightarrow$ N_BASE
N_SFX_BASE+

Each of the leaf nodes in (9) has a lexical entry which provides the c-structure information used to produce the tree and the f(unctional)-structure information used to produce the syntactic LFG analyses. Like standard c-structure rules, sublexical rules and lexical entries can be annotated with information about f-structural information.

For common nouns like *haftA* 'week', no lexical entry is necessary. The Urdu grammar only stipulates lexical entries for nouns which specify an unpredictable subcategorization frame (e.g., nouns used with complex predication). In the case of *haftA* 'week', no such information needs to be stated. The lemma *haftA* is contained in the morphological analyzer and the grammar makes use of this by accessing it via XLE's *-unknown* facility. This facility provides a default lexical entry *-unknown* which can match any lexical item from the morphology. Through it, the morphology itself provides the lexical items, while the *-unknown* lexical entry provides the information used to build the c- and f-structures. A simplified version of *-unknown* for Urdu is shown in (11).

[10]The only way that these rules differ from standard c-structure rules is the presence of _BASE which indicates to XLE that these are *sublexical* rules. XLE uses this sublexical indication for a default collapsed display of the c-structure so that only the N node is shown with the inflected form under it, as shown in (i).

(i)


(11)	-unknown	N	XLE	(↑ PRED)=‘%stem’
				(↑ NTYPE)=common
		NAME	XLE	(↑ PRED)=‘%stem’
				(↑ NTYPE)=proper
		A	XLE	(↑ PRED)=‘%stem’
				(↑ ATYPE)

The %stem matches the form output by the morphology, in this case *haftA* and is passed to the f-structure syntactic represention to create (↑ PRED)=′haftA′ as well as contributing (↑ NTYPE)=common.

Besides the mapping between surface form and abstract tags and lemmata, the morphology-syntax interface allows for a many-to-many mapping between lemmata/tags and functional/syntactic information. Lemmata and tags may be associated with simple or complex functional annotations or not be annotated at all and thus be “suppressed” for the purposes of syntactic analysis.

The major advantage of integrating a finite-state morphology into a grammar, however, is that the lexicons for the grammar can be relatively small. This is because lexicons are not needed for words whose syntactic lexical entry can be determined based on their morphological analysis and for which nothing special need be said (i.e. as described for the *-unknown* entry in (11)). This is particularly true for nouns, adverbs and adjectives. Items whose lexical entry cannot be predicted based on the morphological tags need explicit lexical entries. This is the case for items whose subcategorization frames are not predictable, primarily for verbs. For each verb lemma, a lexical entry is created with the required subcategorization information. Due to the FST morphology, only a single entry is needed, corresponding to that of the lemma; no entry is needed for the inflected forms, thus avoiding the need for multiple entries for forms in morphologically complex languages and simplifying grammar maintenance.

6 Specifying Configurations — MCONFIG

The FSTs used for tokenization, transliteration and morphological analysis can in principle be invoked in different combinations. In the XLE platform the arrangement of the FSTs is specified in the “MCONFIG” file (short for “morphological configuration”) (Crouch et al., 2016). In this section, we show how the Urdu grammar makes use of the power and flexibility afforded by combination of FSTs.

The function of the MCONFIG file is to interface an XLE grammar with external components. Within ParGram, these have generally been FST tokenizers and morphological analyzers. The MCONFIG file is also the place where further transducers can be introduced. For example, the transducer which recognizes Multi-Word Expressions (MWE) and tokenizes these as one token, e.g., for *New York* or *Saudi Arabia*. The Urdu grammar also makes use of a MWE FST (Hautli and Sulger, 2011), as seen in the Urdu MCONFIG file in Figure 4.

```
STANDARD URDU MORPHOLOGY (1.0)

TOKENIZE:
P!TranslitParse.fst G!TranslitGenNoDia.fst

ANALYZE USEFIRST:
morph.override
urdu.fst

PARAMETERS:
*NOCAP

BuildMultiwordsFromMorphology:
Tag = +PreferMWToken
```

Figure 4: Urdu MCONFIG

The first line in the MCONFIG in Figure 4 is an identificatory line. The TOKENIZE section invokes the files relevant for tokenization. It allows for the specification for two different files: one for parsing (indicated by P!) and one for generation (indicated by G!). Since parsing and generation have different goals, they often involve different tokenizers.[11] The Urdu MCONFIG file calls an FST that consists of the tokenizer and the transliterator composed together (section 4). For generation, the version here does not generate diacritics in the Urdu script. However, an alternative version `TranslitGenWithDia.fst` can instead be used to always produce diacritics.

The Urdu grammar also allows for a further option. Rather than taking Urdu script input, it is possible to specify in the MCONFIG file that already-transliterated input be parsed and generated. This is useful for debugging as it separates issues of script from issues of morphological and syntactic analysis. In this case, rather than the custom-made transliterator plus tokenizer, the default tokenizers for parsing and generation available as part of the XLE package are called: `P!/usr/local/xle/bin/default-parse-tokenizer.fsmfile G!/usr/local/xle/bin/default-gen-tokenizer.fst`.

The Urdu morphology is invoked in the `ANALYZE` section with `urdu.fst`. There are two files: the Urdu morphology proper and an FST specificying multiword expressions. The `USEFIRST` option specifies that the first file applies before the Urdu morphology. This has the effect that the FST for multiwords, `morph.override`, is applied first. Furthermore, as the last two

[11]For example, the parser needs to parse errors such as two commas in a row, whereas the generator should not allow for this possibility to be output (see Jurafsky and Martin (2006) for detailed discussions).

lines of the MCONFIG file show, multiword expressions are provided with a tag that allows the multiword version to be preferred (Frank et al., 1998).

Finally, the `PARAMETERS` option allows for the specification of further parameters. The parameter shown here pertains to capitalization. Since neither the Urdu script, nor Devanagari nor the common Latin transliteration scheme need to be normalized for capitalization, this option is set to `NOCAP`.

This section concludes the description of how finite-state morphologies are used in the Urdu grammar. The design decisions laid out in Beesley and Karttunen (2003) for morphological analyzers provide flexibility and formal power while also being well designed for developers. The finite-state morphologies can be integrated into grammar development platforms such as XLE and provide grammar writers with the tools necessary for an efficient and robust morphological analysis. Urdu is morphologically complex, but the formal power in Beesley and Karttunen's finite-state technology allows a straightforward treatment of the morphology and provides a way to separate the morphological issues from the complexities of Urdu-Latin transliteration.

7 Causatives and the Morphology-Syntax Interface

This section describes an issue that arose with respect to the ParGram treatment of complex predication in conjunction with sublexical rules (section 5.2). The issues were observed in both the Turkish (Çetinoğlu, 2009, Çetinoğlu and Oflazer, 2009) and Urdu grammars. Here we illustrate the underlying problem with respect to Urdu.[12]

7.1 Complex Predicates via Restriction

The Urdu grammar uses the *restriction operator* (Kaplan and Wedekind, 1993) to model complex predication. The restriction operator allows for features of f-structures to be "restricted out", i.e., to cause the grammar to function as if these features did not exist. This operation is applied to complex predication in order to build complex predicate-argument structures dynamically (Butt et al., 2003, 2009).

Consider (12). (12) is a noun-verb complex predicate in which *kahAnI* 'story' is an argument that is contributed by the noun *yAd* 'memory', but that functions as the direct object of the clause. The finite verb *kI* 'did' has two arguments: Nadya and *yAd* 'memory'. The noun *yAd* 'memory' thus plays a double role: it is an argument of the finite verb and it contributes to the overall predication of the clause (Mohanan, 1994).

[12]We would like to acknowledge Özlem Çentinoğlu, who was the first to track down the root of the problem in the Turkish grammar.

The details of theoretical and computational analyses of complex predication have been discussed in depth elsewhere (e.g., Butt 1995, Mohanan 1994, Butt et al. 2009). The f-structure analysis is as in (13) (simplified to show only the predicate-argument relations).

(12)

nAdiyah=nE	kahAnI	yAd	k-I
Nadya.F.Sg=Erg	story.F.Sg.Nom	memory.F.Sg.Nom	do-Perf.F.Sg

'Nadya remembered a/the story.'

(13) $$\begin{bmatrix} \textsc{pred} & \text{'do<}\textsc{subj}\text{,'memory<}\textsc{obj}\text{>'>'} \\ \textsc{subj} & \begin{bmatrix} \textsc{pred} & \text{'Nadya'} \end{bmatrix} \\ \textsc{obj} & \begin{bmatrix} \textsc{pred} & \text{'story'} \end{bmatrix} \end{bmatrix}$$

This f-structure is achieved by a dynamic composition of the subcategorization frames contributed by *kar* 'do' and *yAd* 'memory'. The restriction operator is invoked in f-structure annotations on the c-structure rules. While the individual annotations differ according to the type of complex predicate that is involved, (14) illustrates the general schema.

(14) $$\text{Vcp} \rightarrow \begin{array}{cc} \text{X} & \text{Vlight} \\ \downarrow\backslash\text{OldGF}\backslash\textsc{pred}{=}\uparrow\backslash\text{OldGF}\backslash\textsc{pred} & \uparrow{=}\downarrow \\ (\downarrow\text{OldGF}) = (\uparrow\text{NewGF}) & \\ (\uparrow \textsc{pred arg2}){=}(\downarrow \textsc{pred}) & \end{array}$$

To allow the predicate composition, the light verb is associated with a subcategorization frame that is incomplete. In the case of *kar* 'do', the subcategorization frame is as in (15). The %Pred stands for a variable that needs to be filled in, following the XLE notation for variables.

(15) ($\uparrow$ PRED) = 'kar< SUBJ %Pred >'

The *kar* 'do' contributes a subject to the predication and also encodes the fact that it is needs a predicate to fill its variable slot. This further predicate is provided by the X in (14). The X could be a noun, an adjective, a verb or an adposition. In our example, it is the noun *yAd* 'memory'. The annotation ($\uparrow$ PRED ARG2)=($\downarrow$ PRED) substitutes the PRED value of *yAd* into the second argument of the light verb. The subcategorization frame of *yAd* is lexically specified as in (16) and contributes an object to the overall predication.

(16) ($\uparrow$ PRED) = 'yAd<OBJ>'

The restriction operator restricts out those pieces of information which are "changed" as part of complex predication.[13] For one, the overall PRED value changes, since a value has been substituted for the variable. For another, the complex predication may require grammatical functions to be renamed. This does not occur in our example, but does occur with causatives, discussed below. The OldGF stands for a grammatical function, be it a SUBJ, OBJ, OBL OR OBJ_θ that is being reassigned to a NewGF. To accomplish this, it is restricted out and the new correspondence is established via the $(\downarrow \text{OldGF}) = (\uparrow \text{NewGF})$ equation.

The restriction operator is able to model different types of complex predicates in the Urdu grammar and can model cases of stacked complex predicates (Butt et al., 2009).

7.2 Causatives and Passives

Problems with the restriction operator arise in the interaction of passives with causatives. Causatives in Urdu are formed morphologically, as in (17), where the causative morpheme -*A* effectively adds an argument, the causer. (17a) is an example of a simple transitive verb. In the causativized version in (17b),[14] the subject of the transitive is realized as the causee and is marked with the dative/accusative *kO*. This is an example of a "change" in grammatical function that can be effected via the restriction operator.

(17) a.

yassIn=nE	kHAnA	kHa-yA
Yassin=Erg	food.M.Sg.Nom	eat-Perf.M.Sg

'Yassin ate food.'

b.

nAdyA=nE	yassIn=kO	kHAnA	kHil-A-yA
Nadya=Erg	Yassin=Dat	food.M.Sg.Nom	eat-Caus-Perf.M.Sg

'Nadya had Yassin eat (fed Yassin).'

Causatives are also complex predicates since the overall argument structure is co-determined by more than one predicational element. Crosslinguistically, causatives can be either morphological or syntactic but the constraints on the formation of complex predicate-argument structures are the same (Alsina, 1993). Following this insight from theoretical linguistics, the Urdu grammar treats syntactically formed complex predicates like the N-V complex predicates discussed above and morphologically formed causatives on a par. The predicate-argument structure is calculated dynamically based on the information contributed by each of the predicational parts and the

[13]As LFG is formally monotonic, the f-structures are not in fact changed. Rather, new information is added onto already existing information. Pretty printing then gives the effect of an f-structure "changing" into another one.

[14]The stem also changes in this case. The verb *kHA* 'eat' is one of a handful of verbs in Urdu which shows an irregular stem formation in the causative.

final, joint subcategorization frame is effected by the restriction operator. With morphological causatives, this occurs at the level of sublexical rules.

The morphological analyzer provides the analysis in (18) for the verb *kHilAyA* 'made to eat'. As the tags are terminal nodes of sublexical rules, the `+Caus` tag provides a phrase-structure locus for the restriction operator so that it can be analyzed similarly to syntactically formed complex predicates. The sublexical rule for the causative in (17b) is as in (19).

(18) kHilAyA ⇔ kHA +Verb +Caus +Perf +Masc +Sg

$$\text{(19)}\quad \text{V} \rightarrow \quad \begin{array}{c} \text{V-S\_BASE} \\ \downarrow\backslash\textsc{pred}\backslash\textsc{subj}{=}\uparrow\backslash\textsc{pred}\backslash\textsc{subj} \\ (\downarrow\textsc{subj}){=}\ (\uparrow\textsc{obj-go}) \\ (\uparrow \textsc{pred arg2}){=}(\downarrow \textsc{pred}) \end{array} \quad \begin{array}{c} \text{CAUS\_BASE} \\ \uparrow{=}\downarrow \\ \\ \\ \end{array}$$

The causative tag is lexically associated with a subcategorization frame that is parallel to that of the light verb *kar* 'do':

(20) (↑ PRED) = 'cause< SUBJ %Pred >'

The variable is filled by the PRED value of the main verb, the transitive verb *kHA* 'eat'. The basic f-structure for the non-causative version in (17a) is shown in (21). When causativized, the SUBJ is realized as an OBJ-GO. This is the result of the equations in (19) and the resulting complex subcategorization frame is shown in (22).

$$\text{(21)}\quad \left[\begin{array}{ll} \textsc{pred} & \text{'eat<}\textsc{subj,obj}\text{>'} \\ \textsc{subj} & [\ \textsc{pred}\ \text{'Yassin'}\ ] \\ \textsc{obj} & [\ \textsc{pred}\ \text{'food'}\ ] \end{array}\right]$$

$$\text{(22)}\quad \left[\begin{array}{ll} \textsc{pred} & \text{'cause<}\textsc{subj}\text{,'eat<}\textsc{obj-go,obj}\text{>'>} \\ \textsc{subj} & [\ \textsc{pred}\ \text{'Nadya'}\ ] \\ \textsc{obj-go} & [\ \textsc{pred}\ \text{'Yassin'}\ ] \\ \textsc{obj} & [\ \textsc{pred}\ \text{'food'}\ ] \end{array}\right]$$

Now consider the interaction with passivization. Passives in Urdu are formed by combining the verb *jA* 'go' with the perfect form of the main verb. The subject/agent of the base verb is realized as an adjunct and is marked with *se* 'with/from', as shown in (23).

(23) a.

yassIn=nE	kHAnA	kHa-yA
Yassin=Erg	food.M.Sg.Nom	eat-Perf.M.Sg

'Yassin ate food.'

b. kHAnA yassIn=sE kHa-yA ga-yA
food.M.Sg.Nom Yassin=Inst eat-Perf.M.Sg go-Perf.M.Sg
'The food was eaten by Yassin.'

Passivization of the causative in (17b), repeated here in (24a), is as in (24b), where the subject/agent of the causative is realized as an adjunct marked by *se* 'with/from'.

(24) a. nAdyA=**nE** yassIn=kO kHAnA kHil-**A**-yA
Nadya=Erg Yassin=Dat food.M.Sg.Nom eat-**Caus**-Perf.M.Sg
'Nadya had Yassin eat (fed Yassin).'

b. yassIn=kO nAdyA=**sE** kHAnA kHil-**A**-yA
Yassin=Dat Nadya=Inst food.M.Sg.Nom eat-**Caus**-Perf.M.Sg

ga-yA
go-Perf.M.Sg
'The food was fed to Yassin by/through Nadya.'

As originally implemented, the Urdu grammar did not parse examples as in (24b) as grammatical. Moreover, it judged examples as in (25) as grammatical. In this case the former indirect object (OBJ-GO) *Yassin* has been analyzed as a former subject that is now realized as an agentive adjunct.

(25) *nAdyA=**nE** yassIn=sE kHAnA kHil-**A**-yA
Nadya=Erg Yassin=Inst food.M.Sg.Nom eat-**Caus**-Perf.M.Sg

ga-yA
go-Perf.M.Sg
'Nadya made the food be eaten by/through Yassin.'

The underlying problem is architectural. Passivization has traditionally been handled by lexical rules in LFG (Bresnan, 1982). These lexical rules apply in the lexicon directly to the specification of subcategorization frames. For example, the transitive verb *kHA* 'eat' has the subcategorization specification in (26) as part of its lexical entry. (26) states that there is a predicate 'P' (*kHA* in our example) which has a subject and an object and which can optionally undergo passivization. The '@' sign signals a template call Dalrymple et al. (2004) to the template PASS, which effects the passivization via a lexical rule.

```
TRANS (P) = @(PASS (^ pred)='P<(^ SUBJ) (^ OBJ)>').
```

Since this is specified in the lexical entry of *kHA* 'eat', passivization **always** applies before causativization. That is, the lexical rule is applied to the V-S_Base in (19). This is followed by the application of the causativization restriction operator. However, passivization should operate on the entire complex predicate, so that passivization follows causativization. Once this problem was identified, passivization was moved to be part of the sublexical rules via the restriction operator, rather than the lexical rules.

7.3 Summary and Discussion

While the current solution to the passive-causative interaction works, it is not satisfactory. The interaction between causativization and passivization revealed an architectural problem in how argument alternations are treated in the implementation. Within theoretical linguistics, argument alternations are stated at a level of a(rgument)-structure and are independent of particular morphological or syntactic realizations. In the ParGram grammars, passivization continued to be treated via lexical rules, as per classic LFG (but see Wedekind and Ørsnes 2003). One reason for this is that a-structure is not implemented in XLE: predicate arguments are modeled solely via subcategorization frames pertaining to grammatical functions. The interaction between causativization and passives at the morphology-syntax interface highlights that traditional lexical rules do not allow for the right order of application when causativization is morphological but passivization is part of the syntax. This realization was only made possible by the integration of the computational morphological component represented by a finite-state morphology á la Beesley and Karttunen (2003).

8 Conclusion and Discussion

We conclude that an integration of finite-state morphology, as well as tokenizing and in the case of Urdu transliterating finite-state transducers, into a computational grammar taps into a powerful yet intuitive technology that enhances grammar engineering substantially. It is thus a technology that no computational grammar should do without. In the case of the finite-state technology provided by Beesley and Karttunen (2003), the morphologial analysis and the morphology-syntax interface is in line with theoretical ideas on the place of morphology in a modular architecture of grammar (Karttunen, 2003, Dalrymple, 2015).

References

Ahmed, Tafseer, Miriam Butt, Annette Hautli, and Sebastian Sulger. 2012. A Reference Dependency Bank for Analyzing Complex Predicates. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 3145–3151. Istanbul, Turkey: European Language Resources Association (ELRA). ISBN 978-2-9517408-7-7.

Alsina, Alex. 1993. *Predicate Composition: A Theory of Syntactic Function Alternations*. Ph.D. thesis, Stanford University.

Beesley, Kenneth and Lauri Karttunen. 2003. *Finite-State Morphology*. CSLI Publications.

Bögel, Tina. 2012. Urdu-Roman transliteration via finite state transducers. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing (FSMNLP)*.

Bögel, Tina, Miriam Butt, Annette Hautli, and Sebastian Sulger. 2007. Developing a finite-state morphological analyzer for Urdu and Hindi. In T. Hanneforth and K. M. Würzner, eds., *Proceedings of the Sixth International Workshop on Finite-State Methods and Natural Language Processing*, pages 86–96. Potsdam: Potsdam University Press.

Bögel, Tina, Miriam Butt, Annette Hautli, and Sebastian Sulger. 2009. Urdu and the modular architecture of ParGram. In *Proceedings of the Conference on Language and Technology 2009 (CLT09)*. CRULP, Lahore.

Bresnan, Joan, ed. 1982. *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press.

Butt, Miriam. 1995. *The Structure of Complex Predicates in Urdu*. Stanford: CSLI Publications.

Butt, Miriam, Tina Bögel, Annette Hautli, Sebastian Sulger, and Tafseer Ahmed. 2012. Identifying urdu complex predication via bigram extraction. In *In Proceedings of COLING 2012, Technical Papers*, pages 409 – 424. Mumbai, India.

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The Parallel Grammar project. In *Proceedings of COLING 2002: Workshop on Grammar Engineering and Evaluation*.

Butt, Miriam and Tracy Holloway King. 2002. Urdu and the Parallel Grammar project. In *Proceedings of COLING2002, Workshop on Asian Language Resources and International Standardization*, pages 39–45.

Butt, Miriam, Tracy Holloway King, and John T. Maxwell III. 2003. Complex predicates via restriction. In M. Butt and T. H. King, eds., *The Proceedings of the LFG '03 Conference*, pages 92–104. Stanford, CA: CSLI Publications. University at Albany, State University of New York.

Butt, Miriam, Tracy Holloway King, María-Eugenia Niño, and Frédérique Segond. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

Butt, Miriam, Tracy Hollowy King, and Gillian Ramchand. 2009. Complex predication: Who made the child pinch the elephant? In L. Uyechi and L. H. Wee, eds., *Reality Exploration and Discovery: Pattern Interaction in Language and Life*, pages 231–256. Stanford: CSLI Publications.

Çetinoğlu, Özlem. 2009. *A Large Scale LFG Grammar for Turkish*. Ph.D. thesis, Sabanci University.

Çetinoğlu, Özlem and Kemal Oflazer. 2009. Integrating derivational morphology into syntax. In N. N. et al., ed., *Recent Advances in Natural Language Processing V*. Amsterdam: John Benjamins.

Crouch, Dick, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. 2016. Xle documentation. Tech. rep., Palo Alto Research Center, http://ling.uni-konstanz.de/pages/xle/doc/xle_toc.html.

Dalrymple, Mary. 2015. Morphology in the LFG architecture. In M. Butt and T. H. King, eds., *Proceedings of LFG15*, pages 64–83. Stanford: CSLI Publications.

Dalrymple, Mary, Ron Kaplan, and Tracy Holloway King. 2004. Linguistic generalizations over descriptions. In *Proceedings of LFG04*. CSLI On-line Publications.

Forst, Martin and Ron Kaplan. 2006. The importance of precise tokenizing for deep grammars. In *Fifth Language Resource and Evaluation Conference (LREC2006)*.

Frank, Anette. 1999. From parallel grammar development towards machine translation. In *Proceedings of MT Summit VII*, pages 134–142.

Frank, Anette, Tracy Holloway King, Jonas Kuhn, and John Maxwell. 1998. Optimality Theory style constraint ranking in large-scale LFG grammars. In M. Butt and T. H. King, eds., *Proceedings of the LFG98 Conference*. CSLI On-line Publications.

Glassman, Eugene H. 1986. *Spoken Urdu*. Lahore: Nirali Kitaben Publishing House, 6th edn.

Hautli, Annette and Sebastian Sulger. 2011. Extracting and Classifying Urdu Multiword Expressions. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Langauge Technologies (ACL-HLT '11): Student Session*, pages 24–29.

Hautli, Annette, Sebastian Sulger, and Miriam Butt. 2012. Adding an Annotation Layer to the Hindi/Urdu Treebank. *Linguistic Issues in Language Technology* 7(3):1–18s.

Hautli-Janisz, Annette. 2014. *Urdu/Hindi Motion Verbs and Their Implementation in a Lexical Resource*. Ph.D. thesis, University of Konstanz.

Hautli-Janisz, Annette, Tracy Holloway King, and Gillian Ramchand. 2015. Encoding event structure in urdu/hindi verbnet. In *Proceedings of the 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation (NAACL 2015)*, pages 25–33.

Humayoun, Muhammad. 2006. *Urdu Morphology, Orthography and Lexicon Extraction*. Master's thesis, Department of Computing Science, Chalmers University of Technology.

Jurafsky, Daniel and James H. Martin. 2006. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, NJ: Pearson Prentice Hall.

Kaplan, Ron and Jürgen Wedekind. 2000. LFG generation produces context-free languages. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 425—431.

Kaplan, Ronald M. 2005. A method for tokenizing text. In *Festschrift in Honor of Kimmo Koskenniemi's 60th anniversary*. Stanford, CA: CSLI Publications.

Kaplan, Ronald M., John T. Maxwell III, Tracy Holloway King, and Richard Crouch. 2004. Integrating finite-state technology with deep LFG grammars. In *Proceedings of the European Summer School on Logic, Language and Information (ESSLLI) Workshop on Combining Shallow and Deep Processing for NLP*.

Kaplan, Ronald M. and Jürgen Wedekind. 1993. Restriction and correspondence-based translation. In *Proceedings of the 6th Conference of the Association for Computational Linguistics European Chapter (EACL)*, pages 193–202. Utrecht University.

Karttunen, Lauri. 2003. Computing with realizational morphology. In A. Gelbukh, ed., *Computational Linguistics and Intelligent Text Processing*, pages 205–216. Springer Verlag.

Malik, Abbas. 2006. *Hindi Urdu Machine Transliteration System*. Master's thesis, University of Paris.

Malik, Muhammad Kamran, Tafseer Ahmed, Sebastian Sulger, Tina Bögel, Atif Gulzar, Ghulam Raza, Sarmad Hussain, and Miriam Butt. 2010. Transliterating Urdu for a Broad-Coverage Urdu/Hindi LFG Grammar.

In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010)*. European Language Resources Association (ELRA).

Maxwell III, John T. 2006. Efficient generation from packed input. In M. Butt, M. Dalrymple, and T. H. King, eds., *Intelligent Linguistics Architectures: Variations on Themes by Ronald M. Kaplan*, pages 19–34. Stanford: CSLI Publications.

Maxwell III, John T. and Ron Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Lingusitics* 19:571–589.

Mohanan, Tara. 1994. *Argument Structure in Hindi*. CSLI Publications.

Rosén, Victoria, Koenraad De Smedt, Paul Meurer, and Helge Dyvik. 2012a. An Open Infrastructure for Advanced Treebanking. In *META-RESEARCH Workshop on Advanced Treebanking at LREC2012*, pages 22–29. Istanbul, Turkey.

Rosén, Victoria, Paul Meurer, and Koenraad de Smedt. 2009. LFG Parsebanker: A Toolkit for Building and Searching a Treebank as a Parsed Corpus. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories (TLT7)*, pages 127–133. Utrecht: LOT.

Rosén, Victoria, Paul Meurer, Gyri Smørdal Losnegaard, Gunn Inger Lyse, Koenraad De Smedt, Martha Thunes, and Helge Dyvik. 2012b. An integrated web-based treebank annotation system. In I. Hendrickx, S. Kübler, and K. Simov, eds., *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories (TLT11)*, pages 157–167. Lisbon, Portugal: Edições Colibri.

Stewart, Thomas and Gregory Stump. 2006. Paradigm function morphology and the morphology-syntax interface. In G. Ramchand and C. Reiss, eds., *The Oxford Handbook of Linguistic Interfaces*, pages 383–421. Oxford: Oxford University Press.

Sulger, Sebastian, Miriam Butt, Tracy Holloway King, Paul Meurer, Tibor Laczkó, György Rákosi, Cheikh Bamba Dione, Helge Dyvik, Victoria Rosén, Koenraad De Smedt, Agnieszka Patejuk, Ozlem Cetinoglu, I Wayan Arka, and Meladel Mistica. 2013. Pargrambank: The pargram parallel treebank. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 550–560. Sofia, Bulgaria: Association for Computational Linguistics.

Sulger, Sebastian and Ashwini Vaidya. 2014. Towards identifying hindi/urdu noun templates in support of a large-scale lfg grammar. In *Proceedings of the Fifth Workshop on South and Southeast Asian Natural Language*

Processing, pages 1–10. Dublin, Ireland: Association for Computational Linguistics and Dublin City University.

Wedekind, Jürgen and Ron Kaplan. 2012. LFG generation by grammar specialization. *Computational Linguistics* 38(4):867–915.

Wedekind, Jürgen and Bjarne Ørsnes. 2003. Restriction and verbal complexes in LFG: A case study for Danish. In M. Butt and T. H. King, eds., *Proceedings of LFG03*. CSLI On-line Publications.

Zarrieß, Sina, Aoife Cahill, and Jonas Kuhn. 2011. Underspecifying and predicting voice for surface realisation ranking. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1007–1017.