Recent Advances in the Hindi/Urdu LFG Grammar

Miriam Butt, Tina Bögel, Annette Hautli, Tafseer Ahmed, Ghulam Raza and Sebastian Sulger

Workshop on South Asian Syntax and Semantics,
UMASS

March 19th, 2011

Contents

Introduction

ParGram ("Parallel Grammar"): NLP project based on Lexical Functional Grammar (LFG)

- multilingual grammar development project
- large-scale, robust, parallel computational grammars
- so far:
  - larger grammars for English, German, French, Norwegian, Chinese and Japanese
  - smaller grammars for Bahasa Indonesian, Malagasy, Turkish and Welsh
- (recently: ParSem — extends ParGram approach to multilingual, parallel semantics)

Introduction

Possible Applications:

- Machine Translation (made simpler because of deep analysis and parallelism across languages).
- Text Summarization (*parsing* of large corpora, *generation* of summaries)
- Question-Answer Systems (*parsing* of large corpora, *generation* of answers) — successful company built on this (Powerset).

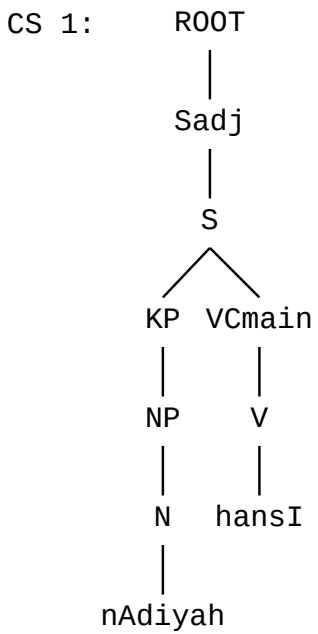
Introduction

Advantages of the ParGram Approach:

- LFG allows for a modular architecture:
  - morphology, syntax and semantics are encoded at independent levels providing necessary flexibility
  - each level of analysis uses different types of representations (e.g., trees vs. AVMs vs. logical formula)
  - all of the levels interact, accounting for interactions across modules
  - an LFG grammar is *reversible*: a grammar can be used for **both** parsing and generation

C- and F-Structure

Example: Analysis from our Urdu grammar for the sentence *naadiyah hansii*

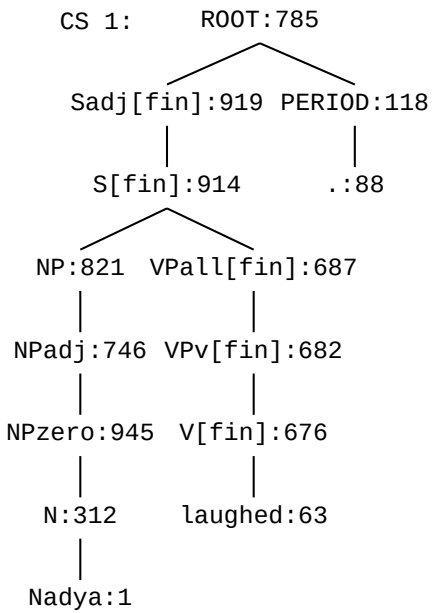


"nAdiyah hansI"

```
[PRED    'hans<[1:nAdiyah]>'
 SUBJ 1  [PRED     'nAdiyah'
          NTYPE    [NSEM [PROPER [PROPER-TYPE name]]
                    NSYN proper]
          SEM-PROP [SPECIFIC +]
          CASE nom, GEND fem, NUM sg, PERS 3]
 CHECK    [_VMORPH [_MTYPE infl]
           _RESTRICTED -, _SUBCAT-FRAME V-SUBJ, _VFORM perf]
 LEX-SEM  [VERB-CLASS unerg]
 TNS-ASP  [ASPECT perf, MOOD indicative]
 CLAUSE-TYPE decl, PASSIVE -, VTYPE main 19]
```

C- and F-Structure

Example: Analysis from the English ParGram grammar for the sentence *Nadya laughed*



"Nadya laughed."

```
⎡PRED     'laugh<[1:Nadya]>'                                                    ⎤
⎢         ⎡PRED  'Nadya'                                                       ⎤⎥
⎢         ⎢CHECK [_LEX-SOURCE morphology, _PROPER known-name]                  ⎥⎥
⎢SUBJ     ⎢NTYPE ⎡NSEM [PROPER [NAME-TYPE first_name, PROPER-TYPE name]]      ⎥⎥
⎢         ⎢      ⎣NSYN proper                                                  ⎥⎥
⎢       1 ⎣CASE nom, GEND-SEM female, HUMAN +, NUM sg, PERS 3                  ⎦⎥
⎢CHECK    [_SUBCAT-FRAME V-SUBJ]                                               ⎥
⎢TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE past]                      ⎥
⎣      63 CLAUSE-TYPE decl, PASSIVE -, VTYPE main                              ⎦
```

ParGram

Advantages of the ParGram Approach:

- The "parallel" in ParGram means:
  - Analyses should abstract away from language particular features as much as possible.
  - A common set of grammatical features is chosen based on common decisions across a range of differing languages.
  - The in-built multilingual perspective means one avoids pitfalls & shortcomings often found in monolingual NLP efforts.

ParGram

Particulars of Grammar Development

- XLE Grammar Development Platform (available by license from PARC)
- integrates: tokenizer (FST), morphological analyzer (FST), syntactic rules (LFG), transfer component (Prolog rewriting rules) used for machine translation and semantic construction
- XLE is written in C; powerful & efficient

Hindi/Urdu grammar at Konstanz

- A small Hindi/Urdu grammar had already been developed at Konstanz.
- Hindi/Urdu is the only South Asian language within ParGram; interesting from a typological point of view.
- **Research Question:** Can the existing small grammar be scaled up to a robust and large-scale grammar within the ParGram context?
- **Some Challenges:**
  - Massive use of complex predicates (about 30% of any text)
  - Free Word-Order, dropping of arguments (problem for generation)
  - Complex interaction between morphology, syntax and semantics (e.g., tense/aspect, case marking, reduplication, Ezafe construction)

Existing Resources

- Much work on some necessary basic resources has been done — most of it at CRULP (fonts, corpora, dictionaries, POS-taggers, etc.)
- Some morphological analyzers have been worked on — however, while in principle these are stand-alone systems, the ParGram context assumes certain types of analyses. So we had to build our own.
- CRULP has worked on an LFG-based Urdu grammar. However, it was designed for parsing, not generation, so did not deal with word order or dropped arguments.

Urdu grammar at Konstanz

Therefore: project on scaling up the existing small Konstanz Hindi/Urdu grammar and developing additional tools.

- 3 years of funding
- official start date: March 1st, 2009
- official end date: February 29th, 2012
- goals: reach broad coverage for Hindi/Urdu grammar (both parsing and generation); develop lexical resources; build semantics module
- this talk: progress in reaching first goal...

The Urdu NP

- progress with respect to NP-internal structure
- word order of Urdu NP is subject to variation
- new NP rule implementation allows for these variations, using two XLE-internal tools: *shuffle operator*, *head precedence operator*

NP Word Order

- valid Urdu NPs; equivalent in meaning:

(1) a. sadr =ko haasil muqaddamaat =se istisnaa
president.M.Sg Dat possessed court-cases.M.Pl Abl immunity.M.Sg
'the immunity from court-cases possessed by the president' rare

b. muqaddamaat =se sadr =ko haasil istisnaa
court-cases.M.Pl Abl president.M.Sg Dat possessed immunity.M.Sg
'the immunity from court-cases possessed by the president' more often

c. sadr =ko muqaddamaat =se haasil istisnaa
president.M.Sg Dat court-cases.M.Pl Abl possessed immunity.M.Sg
'the immunity from court-cases possessed by the president' common

- despite differing word order, internal structure is identical:
  - *haasil* takes *sadr =ko* as argument
  - *sadr =ko haasil* modifies *istisnaa*
  - *istisnaa* takes *muqaddamaat =se* as argument

NP Word Order

Closer look at example (1) c.:

sadr =ko muqaddamaat =se haasil istisnaa
president.M.Sg Dat court-cases.M.Pl Abl possessed immunity.M.Sg
'the immunity from court-cases possessed by the president'

- *sadr ko*, argument of haasil is not adjacent to it
- AP *sadr ko haasil* 'possessed by the president' is a constituent, but its elements are discontinuous
- we find a non-member *muqaddamaat se* between its elements
- arguments of both the noun and its adjective are stacked together irrespective of the logical hierarchical structure

Possible Generalizations

- Generalization #1: Word order is fairly free
- Generalization #2: Arguments must precede their heads

(2) a. *haasil muqaddamaat =se sadr =ko istisnaa
possessed court-cases.M.Pl Abl president.M.Sg Dat immunity.M.Sg

b. *sadr =ko haasil istisnaa muqaddamaat =se
president.M.Sg Dat possessed immunity.M.Sg court-cases.M.Pl Abl

→ **overall, evidence for flat c-structure tree**

→ **at f-structure, associate arguments with their heads based on case/postposition requirements of heads**

Outline of Implementation

Currently, the Urdu NP structure is implemented using two XLE-internal operators:

1. *Shuffle operator:* `[ XP , XP ]`

- items separated by the operator are "shuffled"
- allows the items to occur in any order
- relative order of elements is preserved if no operator is given

```
[A B] , [X Y]

A B X Y
A X Y B
X A Y B
A X B Y
X A B Y
X Y A B
```

Outline of Implementation

Excerpt from NP rule (simplified, without functional annotation):

```
NP -> KP*          e.g.  sadr =ko, muqaddamaat =se
      ,            shuffle operator
      PP*
      ,
      (Aord)
      ,
      AP*          e.g.  haasil (attributive adjectives)
      Nmod         for compounds
      N            e.g.  istisnaa (NP head)
      ,
      (CPrel).
```

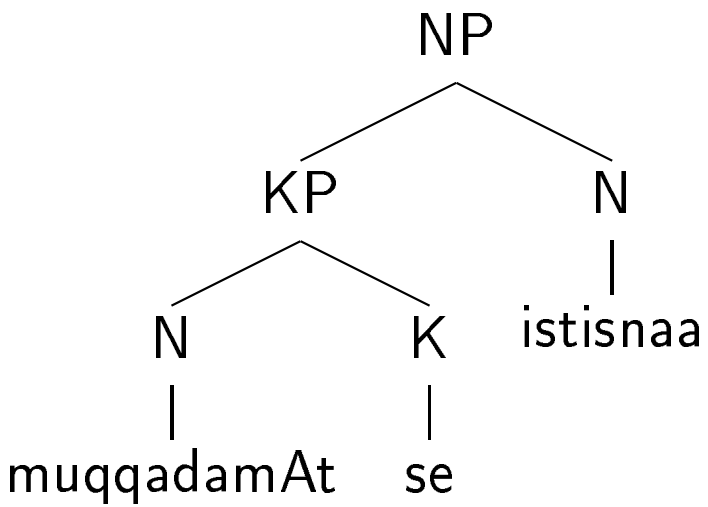
Outline of Implementation

2. *Head precedence operator:* [`f1` >`h` `f2`]

- relation between two (partial) f-structures
- both f-structures must have heads
- head of f2 must precede the head of f1 *in the c-structure tree*
- word associated with head of f2 must precede word associated with head of f1

Outline of Implementation

Example: (ˆ SUBJ) >h (ˆ SUBJ $ OBL)



$$\begin{bmatrix} \text{SUBJ} \begin{bmatrix} \text{PRED } \text{`istisnaa<(}\uparrow \text{ OBL)>'} \\ \text{OBL } \begin{bmatrix} \text{PRED } \text{`muqqadamAt'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

→ head of `OBL` must precede head of `SUBJ`!

Outline of Implementation

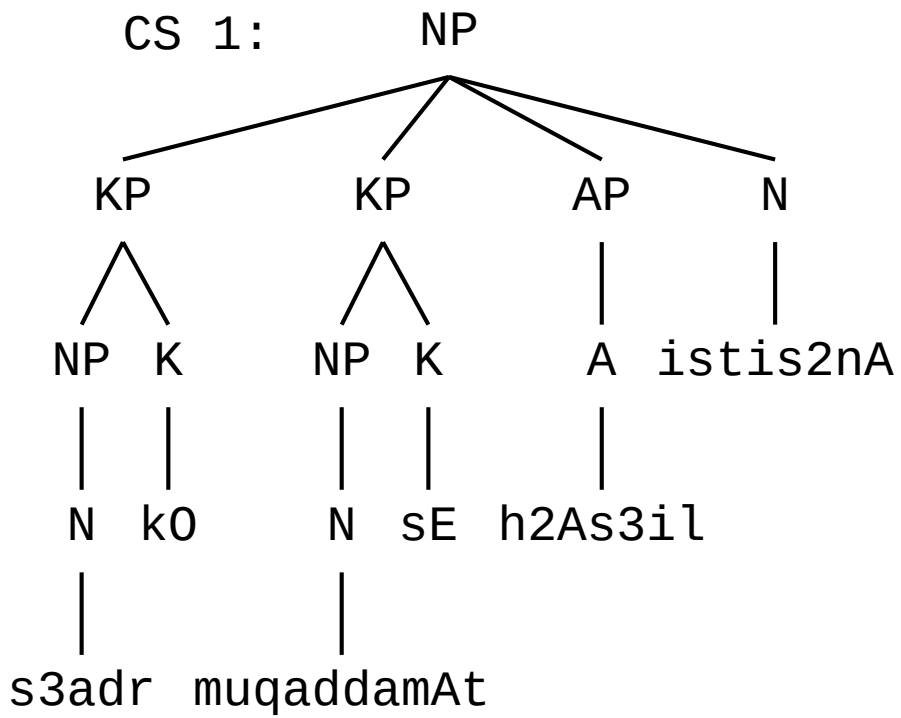
Excerpt from NP rule:

```
NP -> KP*:                                e.g.  sadr =ko
      (^ ADJUNCT $ OBJ-GO)= !
      (^ ADJUNCT) >h (^ ADJUNCT $ OBJ-GO)
      ...
```

- KP is an argument of the adjective *haasil* (which in turn is an ADJUNCT)
- therefore, KP is projected to OBJ-GO function within ADJUNCT set
- head of OBJ-GO must precede head of ADJUNCT in c-structure
- or: *sadr ko* must precede *haasil*

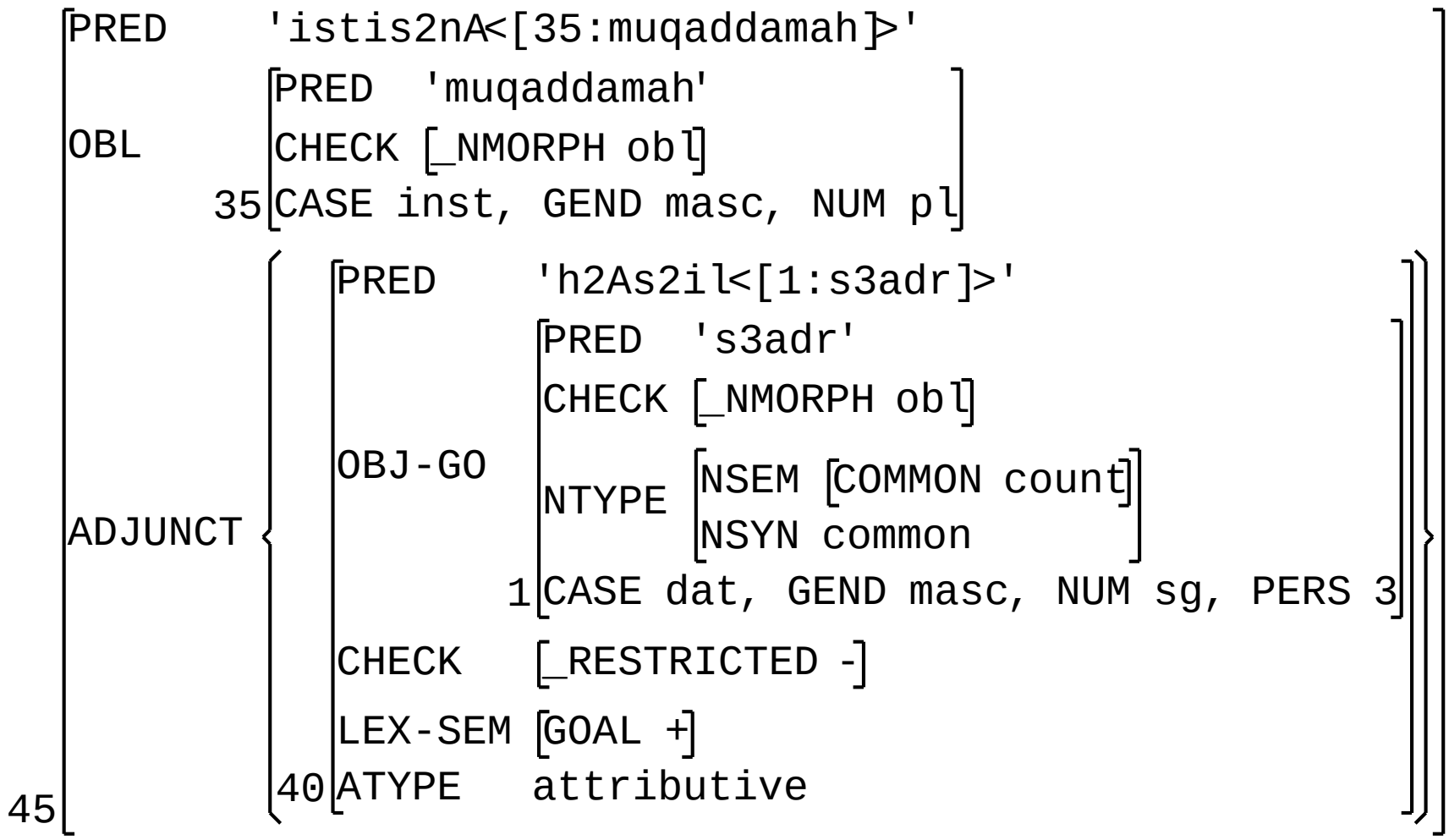
Sample Structures

sadr =ko muqaddamaat =se haasil istisnaa
president.M.Sg Dat court-cases.M.Pl Abl possessed immunity.M.Sg
'the immunity from court-cases possessed by the president'



Sample Structures

"s3adr kO muqaddamAt sE h2As3il istis2nA"



Sample Structures

Note that...

- using the shuffle operator, the NP rule has been designed to allow for varying word order
- using the head precedence operator, the head/argument requirement has been taken care of
- detailed functional annotation ensures that heads select for the correct arguments, based on case/postposition requirements