A quick overview of LFG

April 24, 2004

Contents

1 Short introduction **2**

2 c-structure and f-structure **2**

3 From c-structure to f-structure: regular expressions, unification and lexical entries **2**

3.1 Lexical entries 2

3.2 Step by step, c to f 3

4 Example analysis of two sentences **3**

4.1 Lexical entries 3

4.2 'John made Peter angry' 4

4.3 'Mary gave Jane the book about which John's teacher had said many nice things.' 5

A A very shallow overview of unification **8**

1 Short introduction

LFG, short for Lexical-Functional Grammar, is one of many formal methods of describing grammars of natural languages[1]. As the name implies, the system covers both the *semantics* (Lexical) and the *syntax* (Grammar) by means of connecting them with *functions* (Functional).

An L-F grammar for a language has at least a more or less specified c-structure, an f-structure and a lexicon.

2 c-structure and f-structure

The two central show pieces of LFG are c-structure and f-structure[2]. The c-structure describes the external factors that usually vary by language, while the f-structure tries to capture the common internal structure that is roughly the same everywhere.

*C*onstituent structure describes the exterior form, the order of elements/constituents of the clause. c-structures are regular expressions/trees with the addition of functional schemata placed below each node. The combination of the ordering and the schemata build up the *f*unctional-structure, which describes the interior form, which is not necessarily ordered. The f-structure can be written as an attribute-value matrix (hereafter AVM), or as a list of its defining functions.

3 From c-structure to f-structure: regular expressions, unification and lexical entries

The regular expressions[3] and functional schemata of c-structure build the functions or partial AVMs that, through unification[4], see appendix A, with each other and the lexical entries, generates the full-fledged f-structure.

3.1 Lexical entries

A single lexical entry in LFG consist of a unique reference to the entry (column 1), what c-rule in the c-structure it belongs to (column 2) and a list of functions:

gave:	V	$(\uparrow$ PRED$)$	= 'GIVE⟨SUBJ, OBJ, OBJ2⟩'
		$(\uparrow$ TENSE$)$	= SIMPLEPAST
John:	N	$(\uparrow$ PRED$)$	= 'JOHN'
		$(\uparrow$ NUMBER$)$	= SINGULAR

If there should be another 'gave' in English with a different meaning, the reference and function-list would look different:

gave2:	V	$(\uparrow$ PRED$)$	= 'GIVE⟨SUBJ, OBJ⟩'
		$(\uparrow$ TENSE$)$	= SIMPLEPAST

[1] Others include GPSG, HPSG, minimalism and many others

[2] There are many other ''structures' in LFG, like *s*emantic structure and *a*rgument structure

[3] For the skinny on regular expressions, see Lewis and Papadimitriou (1997) for the theory and any book on the programming language Perl for the practice.

[4] Shown well in Jurafsky and Martin (2000, chapter 11).

3.2 Step by step, c to f

(1) a. Regular expressions with functional schemata...

$$\begin{array}{cccc} S & \rightarrow & NP & VP \\ & & (\uparrow \text{SUBJ}) = \downarrow & \uparrow = \downarrow \end{array}$$

b. ... are equivalent to a tree (a c-structure), and by putting a unique index on each node ...

S_{f_1}

$(\uparrow \text{SUBJ}) = \downarrow$ $\uparrow = \downarrow$

NP_{f_2} VP_{f_4}

c. ... builds functions by replacing the arrows in the functional schemata, ...

$f_1 = f_4$

$(f_1 \text{SUBJ}) = f_2$

d. ... which are equivalent to an attribute-value matrix (AVM), the f-structure.

$$f_1, f_4 \left[\text{SUBJ} \quad f_2 \left[\ldots \right] \right]$$

e. This AVM is then unified with the lexical entries.

$$f_1, f_4 \left[\text{SUBJ} \quad f_2 \begin{bmatrix} \text{PRED} & \text{`J}\textsc{ohn}\text{'} \\ \text{NUMBER} & \textsc{singular} \end{bmatrix} \right]$$

4 Example analysis of two sentences

4.1 Lexical entries

Most nouns and adjectives used below have only PRED for an attribute and will not be listed. The entries for the rest follow:

made: V (↑ PRED) = ‘MAKE⟨SUBJ, OBJ, XCOMP⟩’
(↑ XCOMP SUBJ) = (↑ OBJ)
(↑ TENSE) = SIMPLEPAST

gave: V (↑ PRED) = ‘GIVE⟨SUBJ, OBJ, OBJ2⟩’
(↑ TENSE) = SIMPLEPAST

had said: V (↑ PRED) = ‘SAY⟨SUBJ, OBJ⟩’
(↑ TENSE) = PASTPERFECT

the: D (↑ PRED) = ‘THE’
(↑ SPECTYPE) = DEF

about: P (↑ PRED) = ‘ABOUT⟨OBJ⟩’

which: N (↑ PRED) = ‘PRO’
(↑ PRONTYPE) = REL

John’s: D (↑ PRED) = ‘JOHN’
(↑ SPECTYPE) = POSS

many: D (↑ PRED) = ‘MANY’
(↑ SPECTYPE) = QUANT

things: N (↑ PRED) = ‘THINGS’
(↑ NUM) = PLURAL

4.2 'John made Peter angry'

This first sentence is here interpreted as a causative-construction, not in the 'create'-sense of made. The real problem however is the nature of the XCOMP, as it is a cause of a predicative construction with the copular verb *to be* and not your average verb... I have chosen the solution in Butt et al. (1999, p. 69) but renamed PREDLINK to PREDIC for purely aesthetical reasons.

The c-rules have been simplified to make the c-structure smaller.

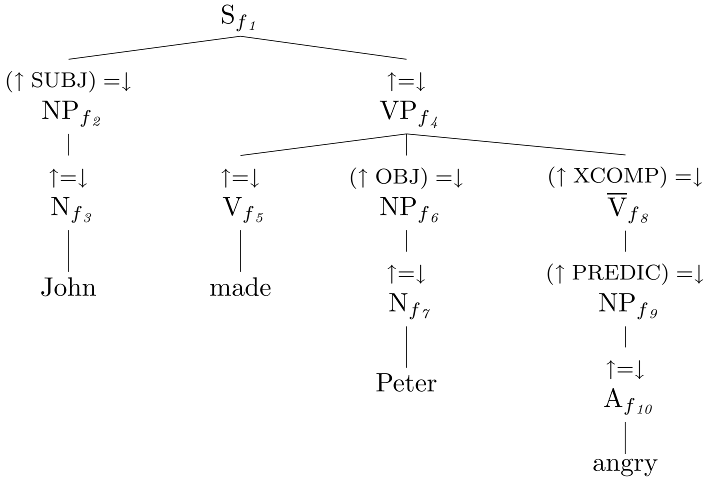
(2) a. S → NP (↑ SUBJ) =↓ VP ↑=↓

b. NP → { A ↑=↓ | N ↑=↓ }

c. VP → V ↑=↓ NP (↑ OBJ) =↓ $\overline{\text{V}}$ (↑ XCOMP) =↓ (↑ XCOMP PRED) = 'be⟨SUBJ, PREDIC⟩'

d. $\overline{\text{V}}$ → NP (↑ PREDIC) =↓

(3) 'John made Peter angry'



$f_1 = f_4 = f_5$
$(f_1\text{SUBJ}) = f_2$
$f_2 = f_3$
$(f_4\text{OBJ}) = f_6$
$f_6 = f_7$
$(f_4\text{XCOMP}) = f_8$
$(f_4\text{XCOMP PREDIC}) = \text{'be}\langle\text{SUBJ, PRED}\rangle\text{'}$
$(f_8\text{PREDIC}) = f_9$
$f_9 = f_{10}$

$$f_1, f_4, f_5 \left[\begin{array}{ll} \text{PRED} & \text{`}\textsc{make}\big\langle\text{SUBJ, XCOMP}\big\rangle\text{'} \\ \text{TENSE} & \textsc{simplepast} \\ \text{SUBJ} & f_2, f_3 \left[\text{PRED} \quad \text{`}\textsc{John}\text{'}\right] \\ \text{OBJ} & f_6, f_7 \left[\text{PRED} \quad \text{`}\textsc{Peter}\text{'}\right] \\ \text{XCOMP} & f_8 \left[\begin{array}{ll} \text{PRED} & \text{`}\textsc{be}\big\langle\text{SUBJ, PREDIC}\big\rangle\text{'} \\ \text{SUBJ} & \\ \text{PREDIC} & f_9, f_{10}\left[\text{PRED} \quad \text{`}\textsc{angry}\text{'}\right]\end{array}\right]\end{array}\right]$$

4.3 ‘Mary gave Jane the book about which John’s teacher had said many nice things.’

The following rules are taken almost verbatim from Dalrymple (2001, chapter 14) and not reproduced here: (28), of $\overline{\text{N}}$, (29), of CP, (31), of RelP, (38), of RTopicPath, and (41), of RelPath. The only difference is that all instances of the symbol ‘RelPro’ has been replaced[5] by the symbol ‘RELATUM’, with equivalent meaning and function.

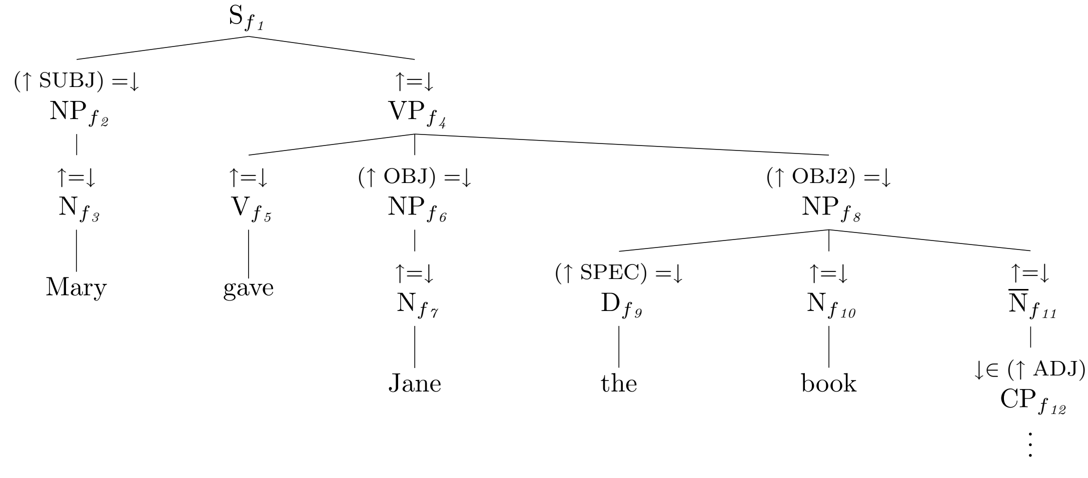
(4) a. $\text{S} \rightarrow \underset{(\uparrow \text{SUBJ}) = \downarrow}{\text{NP}} \quad \underset{\uparrow = \downarrow}{\text{VP}} \quad \left(\underset{(\uparrow \text{ADJ}) = \downarrow}{\text{PP}}\right)$

b. $\text{NP} \rightarrow \left(\underset{(\uparrow \text{SPEC}) = \downarrow}{\text{D}}\right) \left(\underset{(\uparrow \text{ADJ}) = \downarrow}{\text{A}}\right) \quad \underset{\uparrow = \downarrow}{\text{N}} \quad \underset{(\uparrow \text{ADJ}) = \downarrow}{\overline{\text{N}}}$

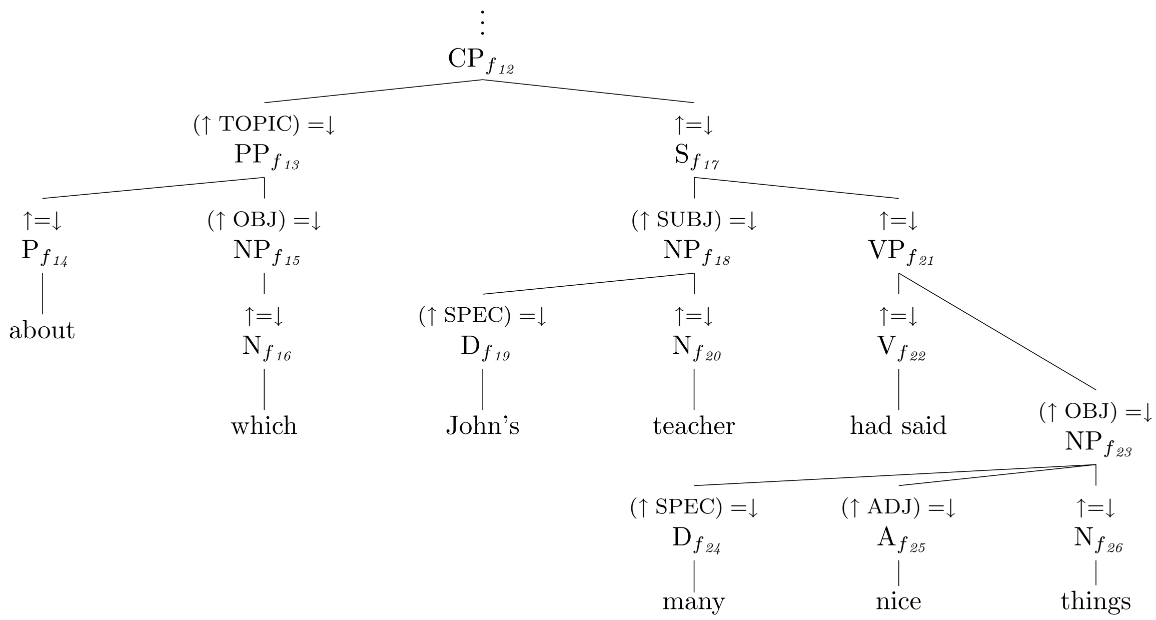
c. $\text{VP} \rightarrow \underset{\uparrow = \downarrow}{\text{V}} \quad \underset{(\uparrow \text{OBJ}) = \downarrow}{\text{NP}} \quad \left(\underset{(\uparrow \text{OBJ2}) = \downarrow}{\text{NP}}\right) \left(\underset{(\uparrow \text{ADJ}) = \downarrow}{\text{PP}}\right)$

d. $\overline{\text{C}} \equiv \underset{\uparrow = \downarrow}{\text{S}}$

(5) ‘Mary gave Jane the book about which John’s teacher had said many nice things.’

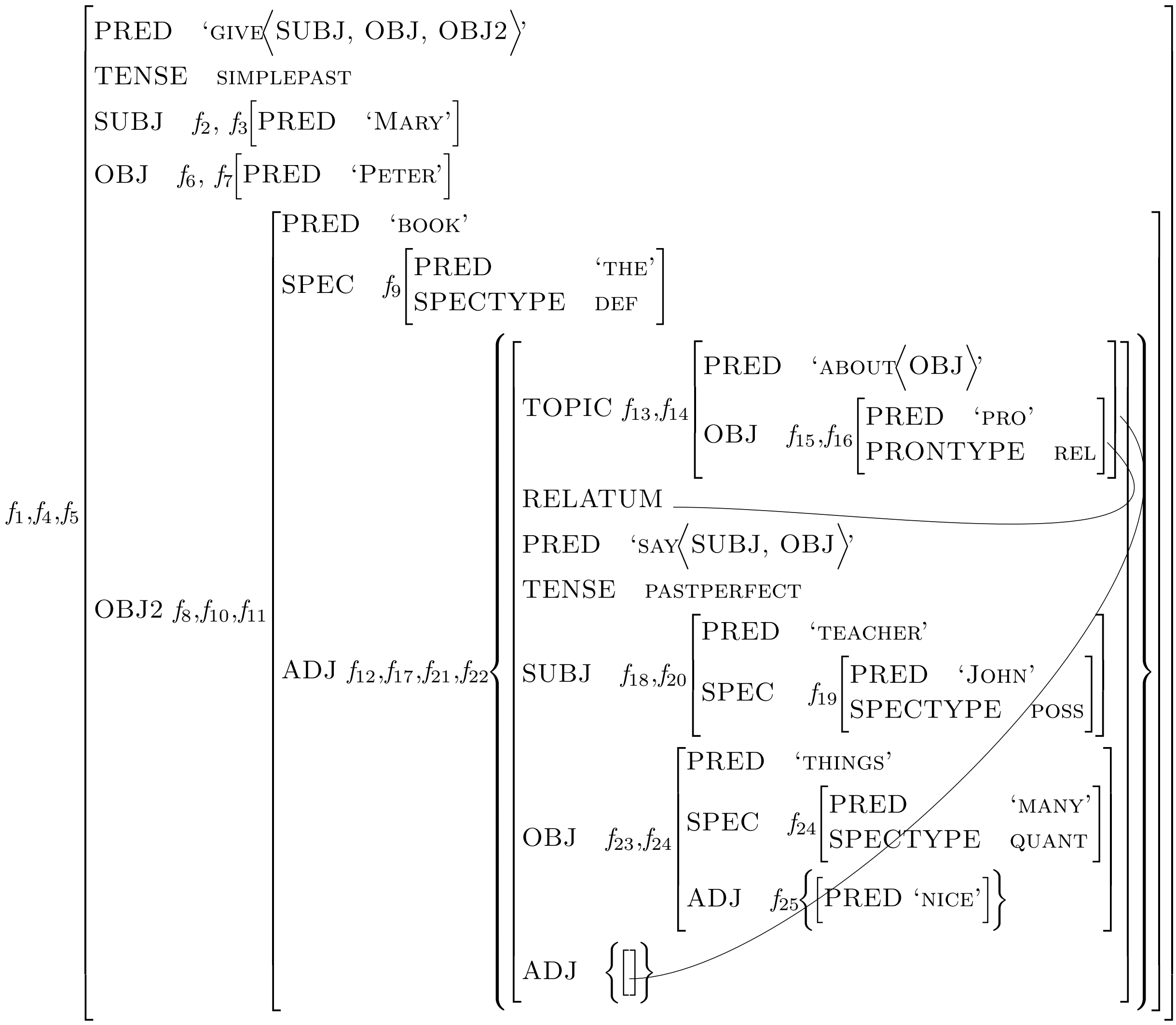


[5] The reason for this is that as many languages does not have relative *pronouns*, a more neutral name for the marker of relativity was needed.



$f_1 = f_4 = f_5$
$(f_1 \text{SUBJ}) = f_2$
$f_2 = f_3$
$(f_4 \text{OBJ}) = f_6$
$f_6 = f_7$
$(f_4 \text{OBJ2}) = f_8$
$(f_8 \text{SPEC}) = f_9$
$f_8 = f_{10} = f_{11}$
$(f_{11} \text{ADJ}) = f_{12}$
$(f_{12} \text{TOPIC}) = f_{13}$
$(f_{12} \text{TOPIC}) = (f_{12} \text{ADJ})$
$(f_{12} \text{RELATUM}) = (f_{12} \text{TOPIC OBJ})$
$(f_{12} \text{RELATUM PRONTYPE}) =_c \textsc{rel}$
$f_{13} = f_{14}$

$(f_{13} \text{OBJ}) = f_{15}$
$f_{15} = f_{16}$
$f_{12} = f_{17}$
$(f_{17} \text{SUBJ}) = f_{18}$
$(f_{18} \text{SPEC}) = f_{19}$
$f_{18} = f_{20}$
$f_{17} = f_{21}$
$f_{21} = f_{22}$
$(f_{21} \text{OBJ}) = f_{23}$
$(f_{23} \text{SPEC}) = f_{24}$
$(f_{23} \text{ADJ}) = f_{25}$
$f_{23} = f_{24}$


PRED ‘GIVE⟨SUBJ, OBJ, OBJ2⟩’
TENSE SIMPLEPAST
SUBJ f2, f3 PRED ‘MARY’
OBJ f6, f7 PRED ‘PETER’
f1,f4,f5
OBJ2 f8,f10,f11
PRED ‘BOOK’
SPEC f9
PRED ‘THE’
SPECTYPE DEF
ADJ f12,f17,f21,f22
TOPIC f13,f14
PRED ‘ABOUT⟨OBJ⟩’
OBJ f15,f16
PRED ‘PRO’
PRONTYPE REL
RELATUM
PRED ‘SAY⟨SUBJ, OBJ⟩’
TENSE PASTPERFECT
SUBJ f18,f20
PRED ‘TEACHER’
SPEC f19
PRED ‘JOHN’
SPECTYPE POSS
OBJ f23,f24
PRED ‘THINGS’
SPEC f24
PRED ‘MANY’
SPECTYPE QUANT
ADJ f25
PRED ‘NICE’
ADJ

A A very shallow overview of unification

'Unification', the verb is 'to unify', is how AVMs are combined into a new AVM. Depending on the AVMs involved, the resulting AVM is either the same size or bigger and more complex than the original AVMs. Point by point:

- An AVM can be empty.
- A non-empty AVM contains one or more *attributes*, each having a *value*.
- The value of an AVM can be another AVM, ergo we get recursion.
- An AVM unifies with an empty AVM.
- An AVM unifies with itself.
- An AVM unifies with any other AVM that it shares no attributes with.
- An AVM unifies with another AVM having the same attributes if the attribute's values are identical, or if AVMs, unify.

References

Miriam Butt, Tracy Holloway King, María-Eugenia Niño, and Frédérique Segond. A grammar writer's cookbook. Number 95 in CSLI lecture notes. CSLI Publications, 1999. ISBN 1575861704.

Mary Dalrymple. Lexical Functional Grammar. volume 34 of *Syntax and semantics*. Academic Press, 2001. ISBN 0126135347.

Daniel Jurafsky and James H. Martin. *Speech and language processing*. Prentice-Hall, Inc., 2000. ISBN 0130950696.

Harry Lewis and Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, 1997. ISBN 0132624788.