Syntactic Formalisms for Parsing Natural Languages

Aleš Horák, Miloš Jakubíček, Vojtěch Kovář
(based on slides by Juyeon Kang)

ia161@nlp.fi.muni.cz

Autumn 2013

Parsing with (L)TAG and LFG

(Lexicalized) Tree Adjoining Grammar (TAG) and Lexical Functional Grammar (LFG)

A) **Same goal**

- formal system to model human speech
- model the syntactic properties of natural language
- syntactic frame work which aims to provide a computaionally precise and psychologically realistic representation of language

B) **Properties**

- Unfication based
- Constraint-based
- Lexicalized grammar

How to parse the sentence in TAG?

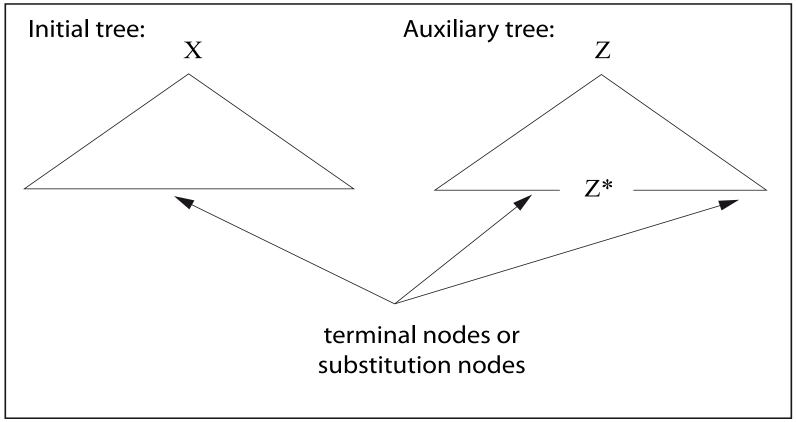
by Joshi, A. Levy, L and Takahashi, M. in 1975

TAG's basic component

- Representation structure: phrase-structure trees
- Finite set of elementary trees
  - Two kinds of elementary trees
    - **Initial trees** (α): trees that can be substituted
    - **Auxiliary trees** (β): trees that can be adjoined

TAG's basic component

- The tree in (X∪Z) are called elementary trees.



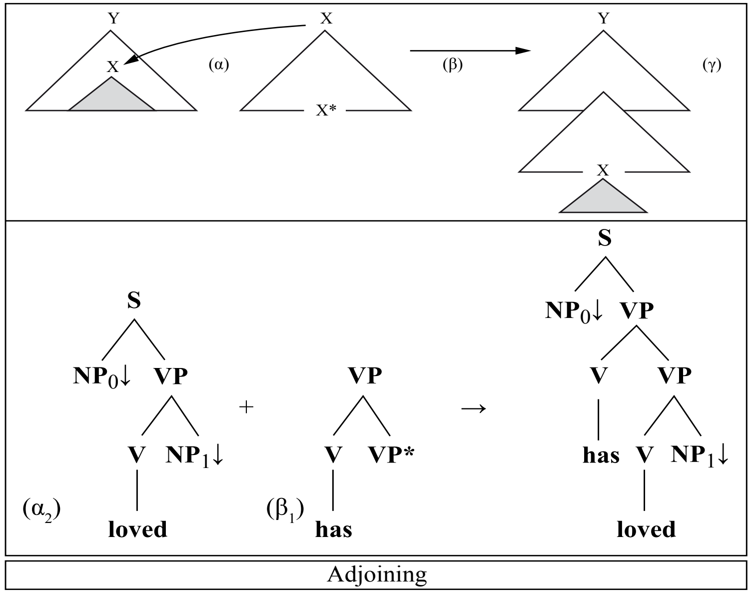
TAG's basic component

- An initial tree (α)
  - all interior nodes are labeled with non-terminal symbols
  - the nodes on the frontier of initial tree are either labeled with terminal symbols, or with non-terminal symbols marked for substitution ($\downarrow$)
- An auxiliary tree (β)
  - one of its frontier nodes must be marked as foot node ($*$)
  - the foot node must be labeled with a non-terminal symbol which is identical to the label of the root node.
- A derived tree (γ)
  - tree built by composition of two other trees
  - the two composition operations that TAG uses adjoining and substitution.

Main operations of combination (1): adjunction

- Sentence of the language of a TAG are derived from the composition of an α and any number of β by this operation.
  - It allows to insert a complete structure into an interior node of another complete structure.
- Three constraints possible
  - Null adjunction (NA)
  - Obligatory adjunction (OA)
  - Selectional adjunction (SA)

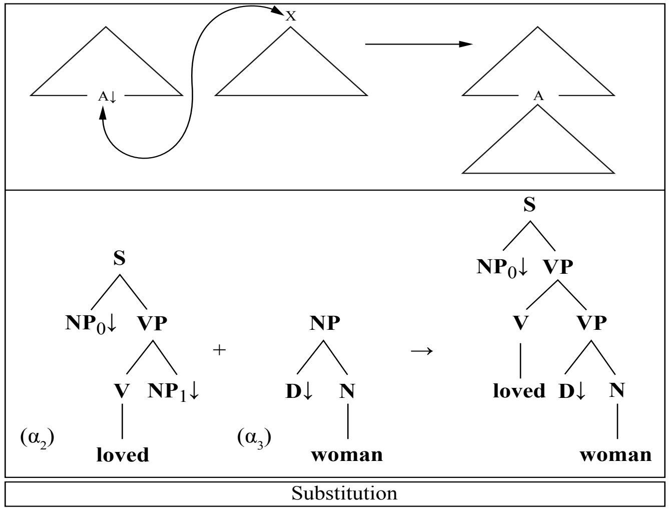
Main operations of combination (1): adjunction



Adjoining

Main operations of combination (2): substitution

- It inserts an initial tree or a lexical tree into an elementary tree.
- One constraint possible
  - Selectional substitution



Substitution

Adjoining constraints

Selective Adjunction ($SA(T)$): only members of a set $T \subseteq A$ can be adjoined on the given node, but the adjunction is not mandatory

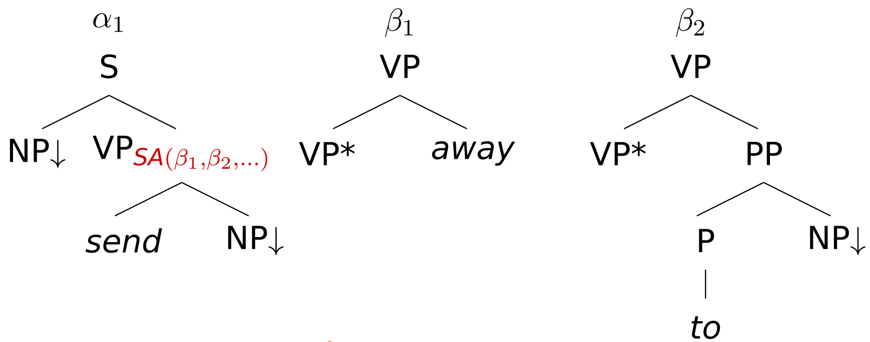
Null Adjunction (NA): any adjunction is disallowed for the given node ($NA = SA(\phi)$)

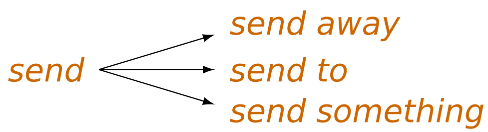
Obligatory Adjunction ($OA(T)$): an auxiliary tree member of the set $T \subseteq A$ must be adjoined on the given node

for short $OA = OA(A)$

Example 1: selective adjunction (SA)

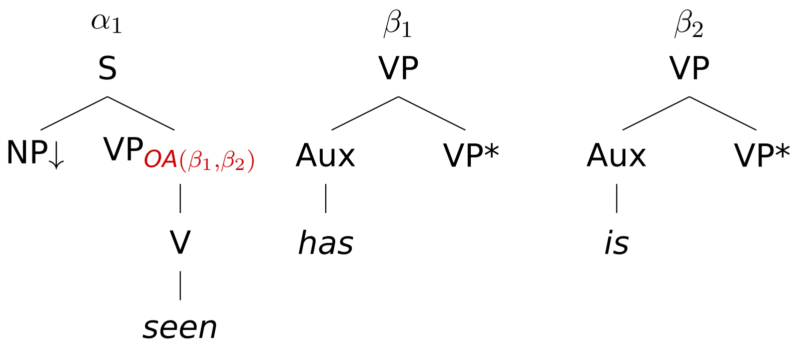
- One possible analysis of "send" could involve selective adjunction:





Example 2: obligatory adjunction

- For when you absolutely must have adjunction at a node:

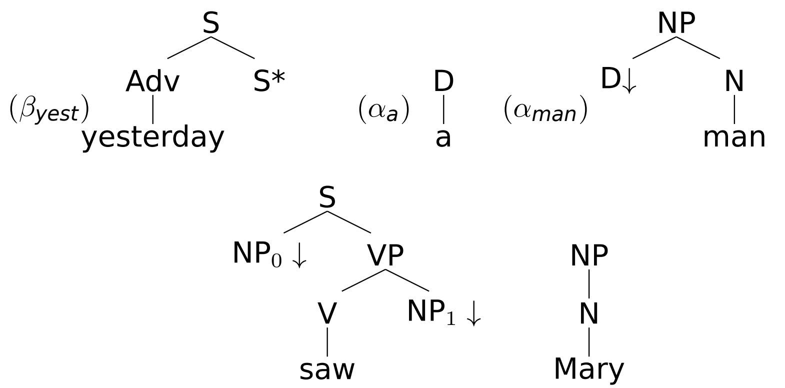


has ⟶ *has seen*

is ⟶ *is seen*

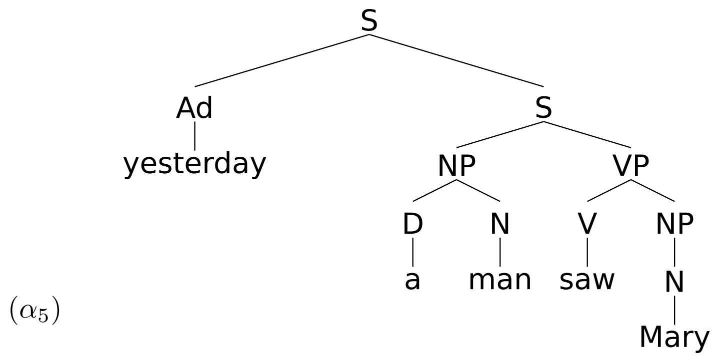
Elementary trees (initial trees and auxiliary trees)

Yesterday a man saw Mary



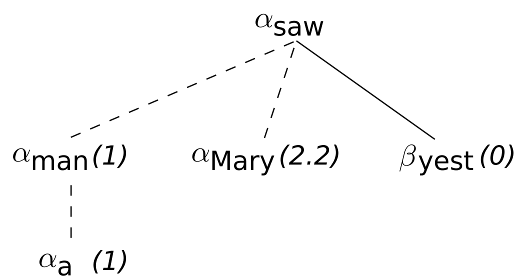
*: foot node/root node
↓: substitution node

Elementary trees (initial trees and auxiliary trees)

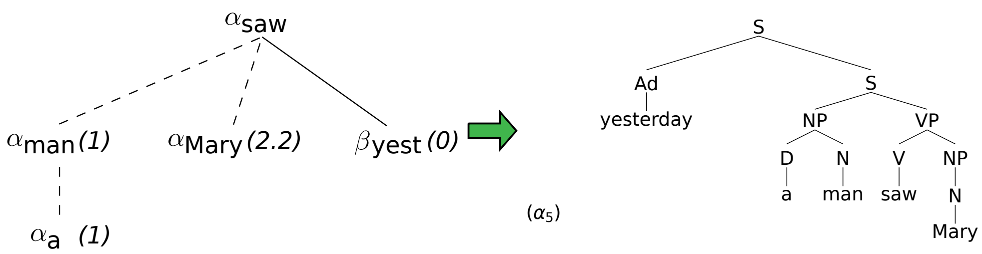


Derivation tree

- Specifies how a derived tree was constructed
  - The root node is labeled by an S-type initial tree.
  - Other nodes are labeled by auxiliary trees in the case of adjoining or initial trees in the case of substitution.
  - A tree address of the parent tree is associated with each node.



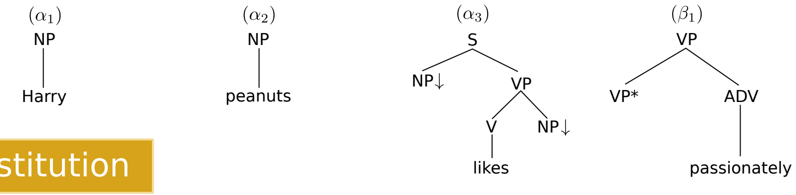
Derivation tree and derived tree α_5



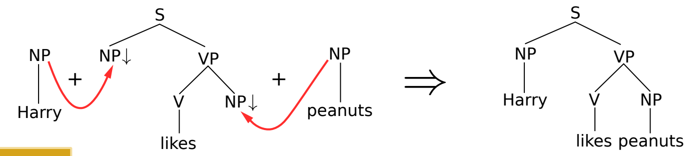
- - - - : substitution operation
______ : adjunction operation

Example 1: Harry likes peanuts passionately

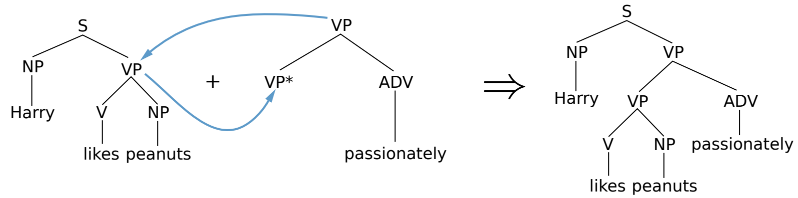
Step 1



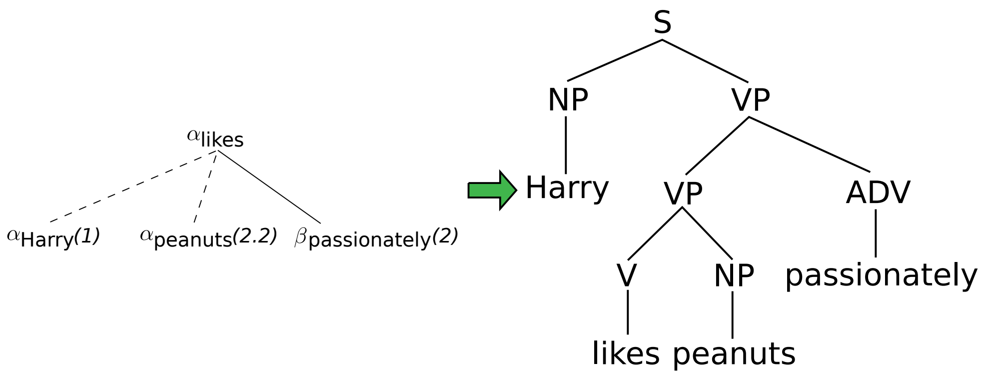
Step 2: substitution



Step 3: adjunction



Derivation tree and derived tree of Harry likes peanuts passionately

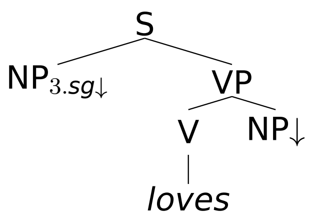


Two important properties of TAG

- Elementary trees can be of arbitrary size, so the domain of locality is increased
  - Extended domain of locality (EDL)
- Small initial trees can have multiple adjunctions inserted within them, so what are normally considered non-local phenomena are treated locally
  - Factoring recursion from the domain of dependency (FRD)

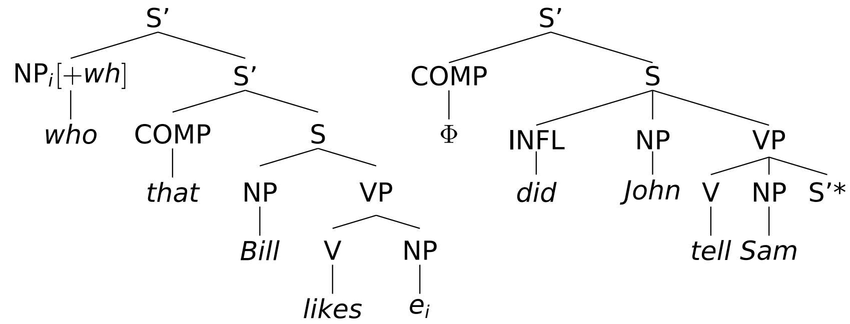
Extended domain of locality (EDL): Agreement

- The lexical entry for a verb like "loves" will contain a tree like the following:



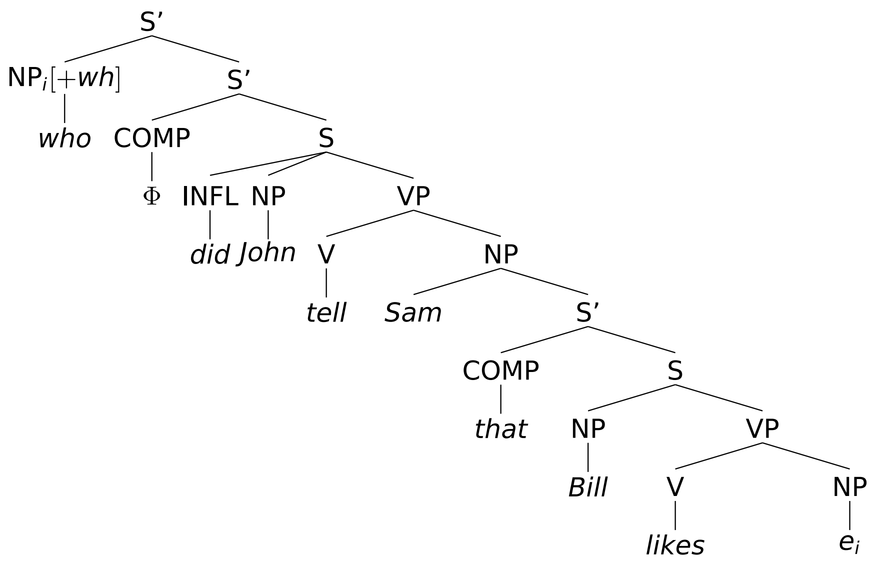
With EDL, we can easily state agreement between the subject and the verb in a lexical entry

Factoring recursion from the domain of dependency (FRD): Extraction



The above trees for the sentence "who did John tell Sam that Bill likes ?" allow the insertion of the auxiliary tree in between the WH-phrase and its extraction site, resulting a long distance dependency; yet this is factored out from the domain of locality in TAG.

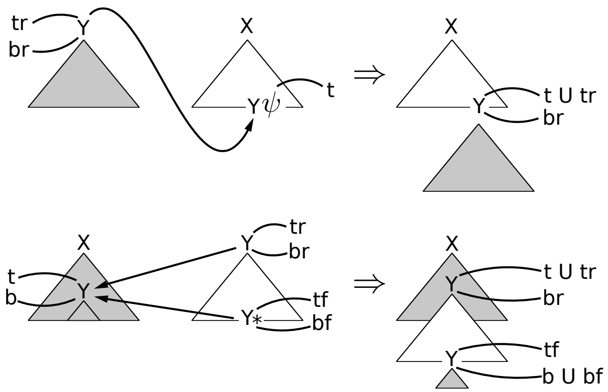
Factoring recursion from the domain of dependency (FRD): Extraction



Variations of TAG

- Feature Structure Based TAG (FTAG: Joshi and Shanker, 1988)

each of the nodes of an elementary tree is associated with two feature structures:
top & bottom Substitution



Substitution with features

Adjoining with features

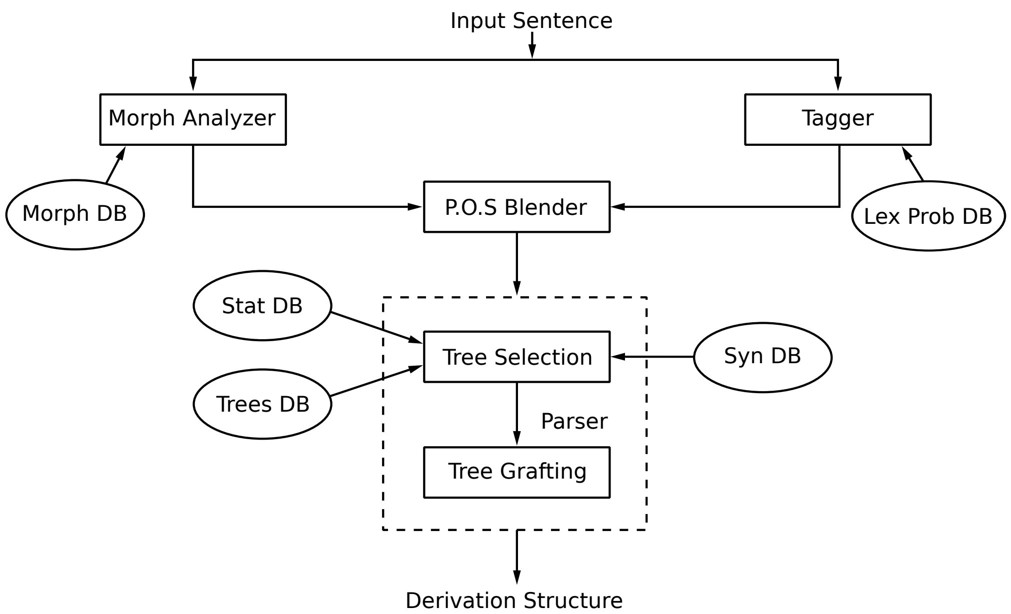
Variations of TAG

- Synchronous TAG (STAG: Shieber and Schabes, 1990)
  - A pair of TAGs characterize correspondences between languages
  - Semantic interpretation, language generation and translation
- Muti-component TAG (MCTAG: Chen-Main and Joshi, 2007)
  - A set of auxiliary tree can be adjoined to a given elementary tree
- Probabilistic TAG (PTAG: Resnik, 1992, Shieber, 2007)
  - Associating a probability with each elementary tree
  - Compute the probability of a derivation

XTAG Project (UPenn, since 1987 ongoing)

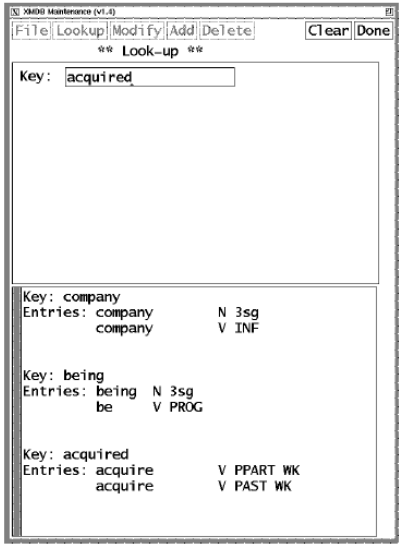
- A long-term project to develop a wide-coverage grammar for English using the Lexicalized Tree-Adjoining Grammar (LTAG) formalism
- Provides a grammar engineering platform consisting of a parser, a grammar development interface, and a morphological analyzer
- The project extends to variants of the formalism, and languages other than English

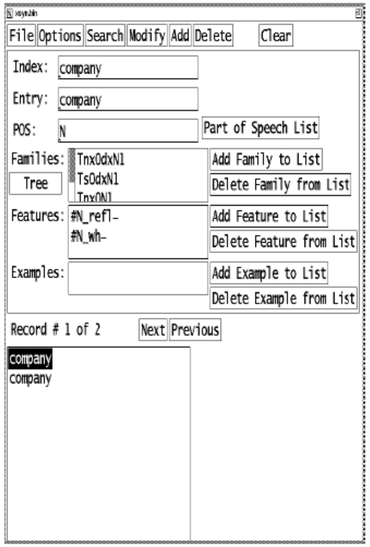
XTAG system



Components in XTAG system

- Morphological Analyzer & Morph DB: 317K inflected items derived from over 90K stems
- POS Tagger & Lex Prob DB: Wall Street Journal-trained 3-gram tagger with N-best POS sequences
- Syntactic DB: over 30K entries, each consisting of:
  - Uninflected form of the word
  - POS
  - List of trees or tree-families associated with the word
  - List of feature equations
- Tree DB: 1004 trees, divided into 53 tree families and 221 individual trees





(a) Morphology database (b) syntactic database

Interfaces to the database maintenance tools

Interface to the XTAG system

Parser evaluation in XTAG Project by [Bangalore,S. *et.al*, 1998]
`http://www.cis.upenn.edu/~xtag/`

How to parse the sentence in LFG?

by Bresnan, J. and Kaplan, R.M. In 1982

Main representation structures

- ***c-structure***: constituent structure

level where the surface syntactic form, including categorical information, word order and phrasal grouping of constituents, is encoded.

- ***f-structure***: functional structure

internal structure of language where grammatical relations are represented. It is largely invariable across languages. (e.g. SUBJ, OBJ, OBL, (X)COMP, (X)ADJ)

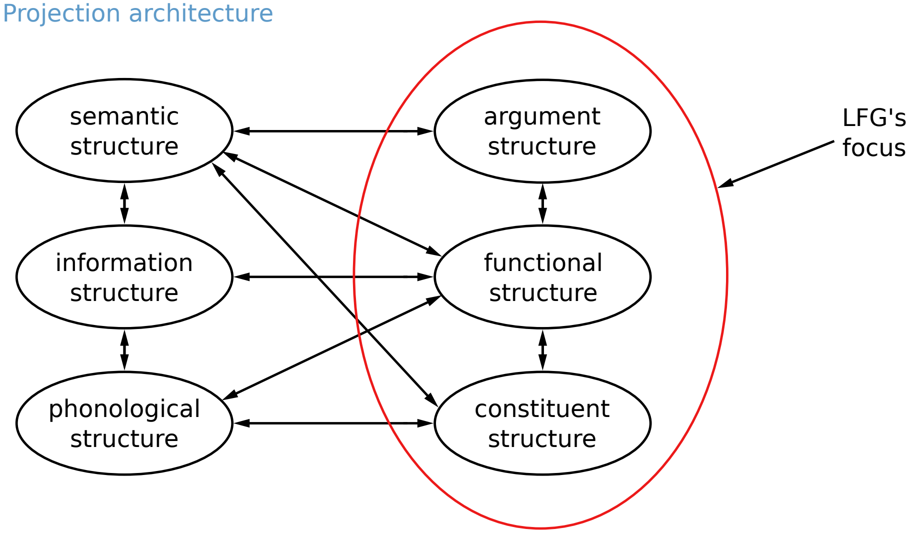
- ***a-structure***: argument structure

They encode the number, type and semantic roles of the arguments of a predicate.

Level of structures and their interaction in LFG

Functional

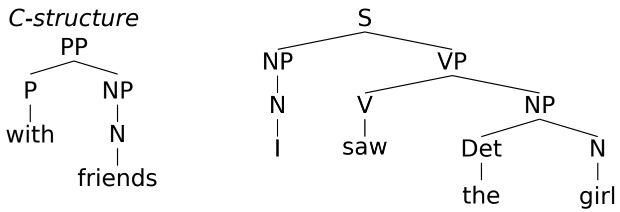
Projection architecture



Level of structures and their interaction in LFG

- In LFG, the parsing result is grammatically correct only if it satisfies 2 criteria:
  1. the grammar must be able to assign a correct c-structure
  2. the grammar must be able to assign a correct well-formed f-structure

c-structure



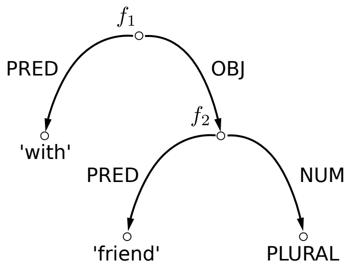
- The constituent structure represents the organization of overt phrasal syntax
- It provides the basis for phonological interpretation
- Languages are very different on the c-structure level :external factors that usually vary by language

Properties of c-structure

- *c-structures* are conventional phrase structure trees:

  they are defined in terms of syntactic categories, terminal nodes, dominance and precedence.
- They are determined by a context free grammar that describes all possible surface strings of the language.
- LFG does not reserve constituent structure positions for affixes: all leaves are individual words.

f-structure



- Attribute-Value notation for *f-structure*

$$\begin{bmatrix} \text{PRED} & \text{'with'} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{'friend'} \\ \text{NUM} & \text{PLURAL} \end{bmatrix} \end{bmatrix}$$

1. representation of the functional structure of a sentence
2. *f-structure* match with *c-structure*
3. it has to satisfy three formal constraints: consistency, coherence, completeness
4. language are similar on this level: allow to explain cross-linguistic properties of phenomena

Examples of *f-structure*

1

OBJ	PRED	'Veit'
TENSE	PAST	
PRED	'send⟨SUBJ, OBJ, OBJ2⟩'	
SUBJ	PRED	'Sabine'
OBJ2	PRED	'e-mail'
	DEF	-
	NUM	SG

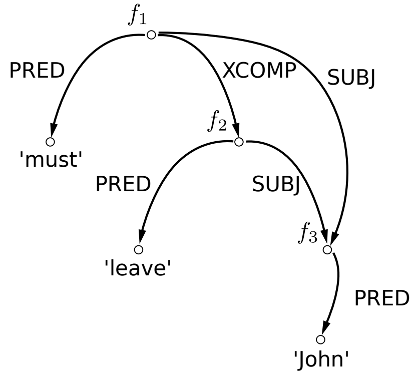
2

SUBJ	PRED	'teacher'	
	DEF	+	
	NUM	SG	
TENSE	PAST		
PRED	'insist ⟨SUBJ, OBL_{on} OBJ⟩'		
OBL_{on}	PCASE	OBL_{on}	
	OBJ	PRED	'homework'
		DEF	+
		NUM	SG

Constraint 1: *f-structure* must be consistent

1. Two paths in the graph structure may designate the same element-called unification, structure-sharing

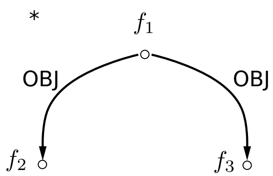
Ex: *John must leave*



PRED	'must'		
SUBJ	PRED	'John'	
XCOMP	PRED	'leave'	
	SUBJ		

Constraint 1: *f-structure* must be consistent

2. attributes are functionally unique - there may not be two arcs with the same attribute from the same f-structure



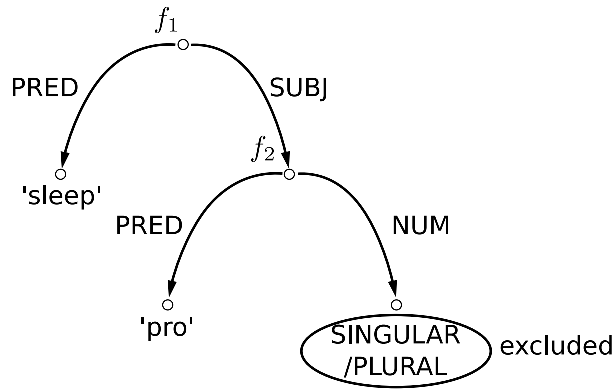
Incosnistent f-structure

SUBJ	PRED	'Veit'
SUBJ	PRED	'Tom'
PRED	'sleep⟨(↑SUBJ)⟩'	
TENSE	PAST	
TENSE	FUT	

Constraint 1: *f-structure* must be consistent

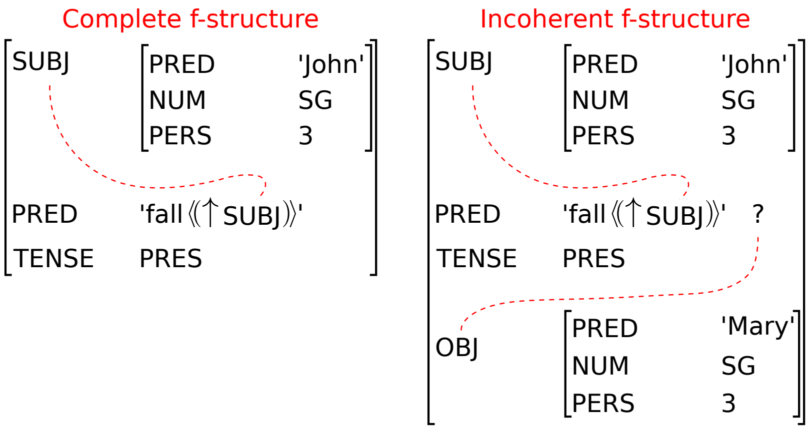
3. The symbols used for atomic f-structure are distinct - it is impossible to have two names for a single atomic f-structure ("clash")

*They sleeps



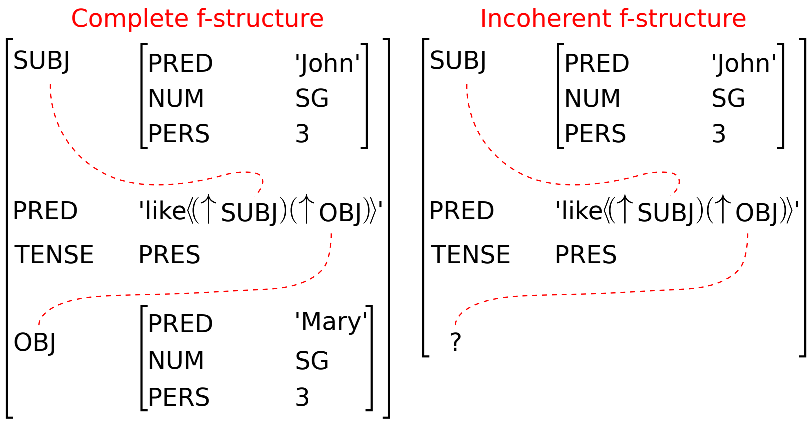
Constraint 2: *f-structure* must be coherent

All argument functions in an *f-structure* must be selected by the local PRED feature.

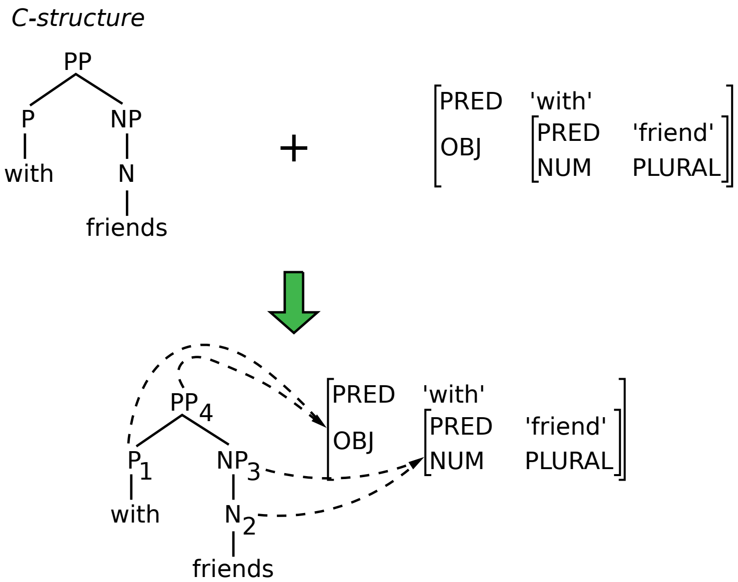


Constraint 3: *f-structure* must be complete

All functions specified in the value of a PRED feature must be present in the *f-structure* of that PRED.



Correspondence between different levels in LFG

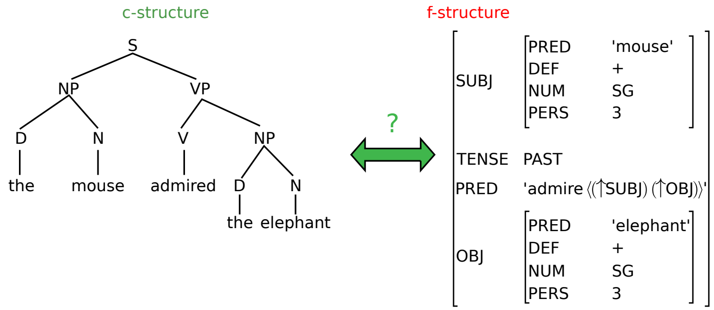


Structural correspondence

- *c-structures* and *f-structures* represent different properties of an utterance
- How can these structures be associated properly to a particular sentence?
- Words and their ordering carry information about the linguistic dependencies in thesentence
- This is represented by the *c-structure* (licensed by a CFG)
- LFG proposes simple mechanisms that maps between elements from one structure and those of another: correspondence functions
- A function allows to map c-structures to f-structures $\Phi : N \rightarrow F$

Mapping the c-structure into the f-structure

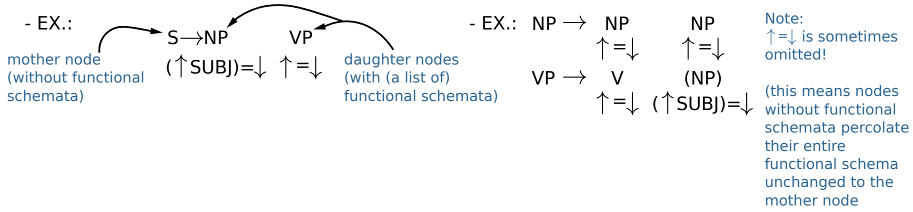
- Since there is no isomorphic relationship between structure and function LFG assumes *c-structure* and *f-structure*
- The mapping between *c-structure* and *f-structure* is the core of LFG's descriptive power
- The mapping between *c-structure* and *f-structure* is located in the grammar (PS) rules



Mapping mechanism: 6 steps

STEP 1: PS rules

- Context-free phrase structure rules
- Annotated with functional schemata



Mapping mechanism: 6 steps

STEP 2: Lexicon entries

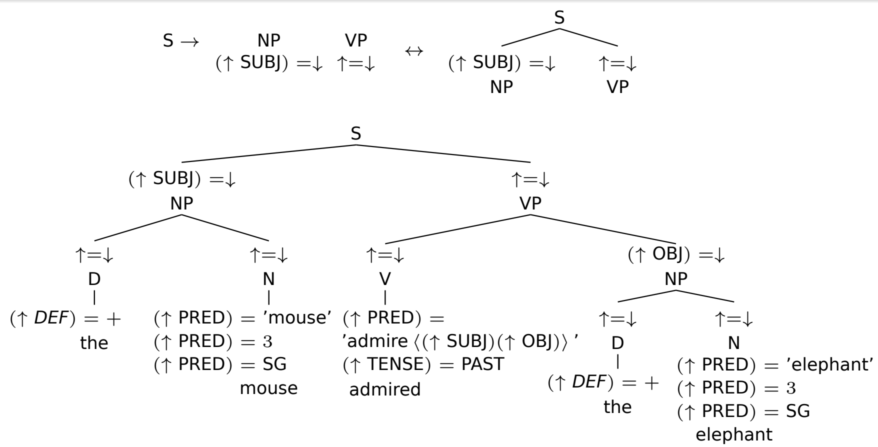
- Lexicon entries consists of three parts: representation of the word, syntactic category, list of functional schemata

Ex.:	mouse	N	(↑PRED)='mouse'
			(↑PERS)=3
			(↑NUM)=SG
	the	D	(↑DEF)=+
	admire	V	(↑PRED)='admire ⟨(↑ SUBJ)(↑ OBJ)⟩'
	-ed	Aff	(↑TENSE)=PAST

Mapping mechanism: 6 steps

STEP 3: *c-structure*

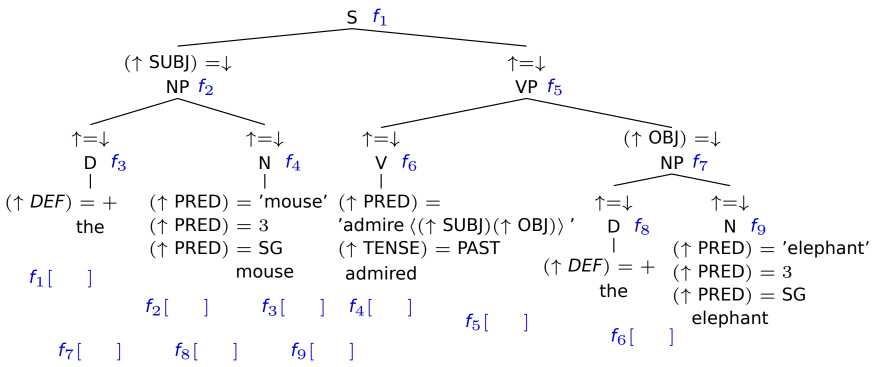
- Like the PS rules, each node in the tree is associated with a functional schemata
- With the functional schemata of the lexical entries at the leaves we obtain a complete *c-structure*



Mapping mechanism: 6 steps

STEP 4: Co-indexation

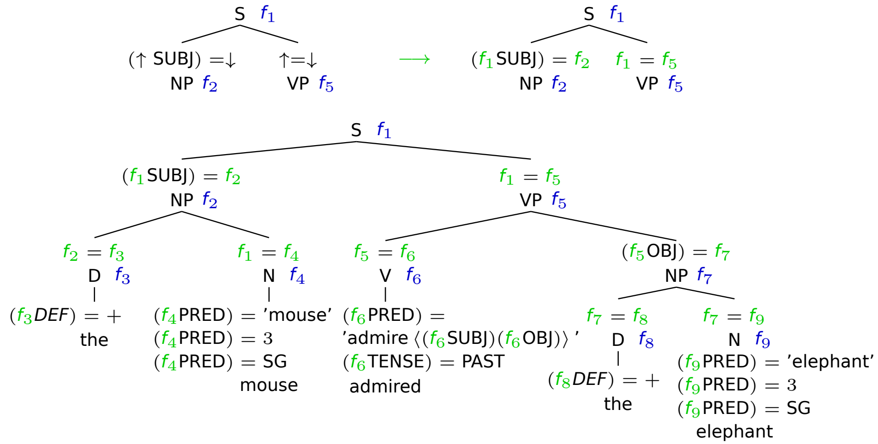
- An f-structure is assigned to each node of the c-structure
- Each of these f-structures obtains a name ($f_1 - f_n$)
- Nodes in the c-structure and associated f-structure are co-indexed, i.e. obtain the same name
- F-structure names $f_1 - f_n$ can be chosen freely but they may not occur twice



Mapping mechanism: 6 steps

STEP 5: Metavariable biding

- All meta-variables are replaced by the names of the *f-structure* representation



Mapping mechanism: 6 steps

- We introduce at this point the notion of **functional equation**
- By listing all functional equations from a *c-structure* we obtain the functional description, called **f-description**

$(f_1\text{SUBJ}) = f_2$	$(f_6\text{PRED}) = \text{'admire}\langle(f_6\text{SUBJ})(f_6\text{OBJ})\rangle\text{'}$
$f_2 = f_3$	$(f_6\text{TENSE}) = \text{PAST}$
$(f_3\text{DEF}) = +$	$(f_5\text{OBJ}) = f_7$
$f_2 = f_4$	$f_7 = f_8$
$(f_4\text{PRED}) = \text{'mouse'}$	$(f_8\text{DEF}) = +$
$(f_4\text{PERS}) = 3$	$f_7 = f_9$
$(f_4\text{NUM}) = \text{SG}$	$(f_9\text{PRED}) = \text{'elephant'}$
$f_1 = f_5$	$(f_9\text{PERS}) = 3$
$f_5 = f_6$	$(f_9\text{NUM}) = \text{SG}$

Table : f-description

Mapping mechanism: 6 steps

STEP 6: From f-description to *f-structure*

- Computation of an *f-structure* is based on the **f-description**
- For the derivation of *f-structures* from the **f-description** it is important that no information is lost and that no information will be added
- The derivation is done by the application of the **functional equations**

List of functional equations

a) simple equations of the form: $f_nA) = B$

b) f-equations of the form: $f_n = f_m$

c) f-equations of the form: $f_nA) = f_m$

→ Functional equations with the same name are grouped into an *f-structure* of the same name

Application of the functional equation (a): $(f_n A) = B$

$(f_3\text{DEF})=+$
$(f_4\text{PRED})=\text{'mouse'}$
$(f_4\text{PERS})=3$
$(f_4\text{NUM})=\text{SG}$
$(f_6\text{PRED})=\text{'admire}\langle(f_6\text{SUBJ})(f_6\text{OBJ})\rangle\text{'}$
$(f_6\text{TENSE})=\text{PAST}$
$(f_8\text{DEF})=+$
$(f_9\text{PRED})=\text{'ELEPHANT'}$
$(f_9\text{PERS})=3$
$(f_9\text{NUM})=\text{SG}$

f_4 [PRED 'mouse' / PERS 3 / NUM SG]
f_3 [DEF +]
f_6 [PRED 'admire $\langle(f_6\text{SUBJ})(f_6\text{OBJ})\rangle$' / TENSE PAST]
f_8 [DEF +]
f_9 [PRED 'elephant' / PERS 3 / NUM SG]

Application of the functional equation (b): $f_n = f_m$

$f_2 = f_3$
$f_2 = f_4$
$f_1 = f_5$
$f_5 = f_6$
$f_7 = f_8$
$f_7 = f_9$

$f_1 f_5 f_6$ [PRED 'admire $\langle(f_6\text{SUBJ})(f_6\text{OBJ})\rangle$' / TENSE PAST]

$f_2 f_3 f_4$ [PRED 'mouse' / PERS 3 / NUM SG / DEF +] ← unification — $f_2 f_3$ [DEF +]

$f_7 f_8$ [DEF +] — unification → $f_7 f_8 f_9$ [PRED 'elephant' / PERS 3 / NUM SG / DEF +]

Application of the functional equation (c): $(f_n A) = f_m$

$(f_1\,\text{SUBJ})=f_2$
$(f_5\,\text{OBJ})=f_7$

$f_1 f_5 f_6$ [PRED 'admire $\langle(f_6\text{SUBJ})(f_6\text{OBJ})\rangle$' / TENSE PAST / SUBJ $f_2 f_3 f_4$[PRED 'mouse' / PERS 3 / NUM SG / DEF +] / OBJ $f_7 f_8 f_9$[PRED 'elephant' / PERS 3 / NUM SG / DEF +]]

unification: $f_2 f_3 f_4$ [PRED 'mouse' / PERS 3 / NUM SG / DEF +]

unification: $f_7 f_8 f_9$ [PRED 'elephant' / PERS 3 / NUM SG / DEF +]

STEP 1: lexical entries

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

STEP 2: c-structure

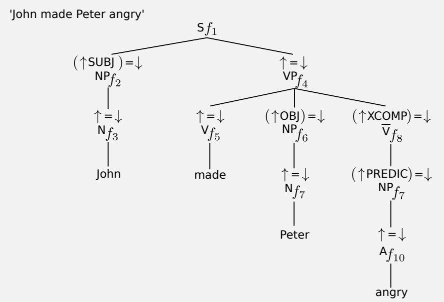
a. S → NP (↑ SUBJ) =↓ VP ↑=↓

b. NP → { A ↑=↓ | N ↑=↓ }

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

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

STEP 3: f-structure



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

STEP 4: unification

f_1, f_4, f_5 [PRED 'make ⟨SUBJ, XCOMP⟩ ' / TENSE simplepast / SUBJ f_2, f_3[PRED 'John'] / OBJ f_6, f_7[PRED 'Peter'] / XCOMP f_8[PRED 'be ⟨SUBJ, PRED⟩ ' / SUBJ (→ OBJ) / PREDIC f_9, f_{10}[PRED 'angry']]]