

# Comparative Introduction to Lexicalist Syntactic Theories

**Dan Flickinger**

**On building a more efficient grammar by  
exploiting types**

Emilia Ellsiepen  
millaL7@web.de

# Outline

- Grammar engineering and systems
- Eliminating disjunctive features
- Reducing number of lexical entries
- Reducing size of feature structures
- Summary

# Grammar engineering I

Task of the grammar engineer:

- balancing clarity and efficiency in grammar description

Tools:

- provided by platforms
- may overlap in functionality

# Grammar engineering II

## Efficiency

- is not regarded as a typical subject to grammar engineers
- is usually optimized in parsing or generation, with no regard to the design of the grammar itself

The aim of the experiments discussed here:

- try to optimize efficiency without changing the engines to run the grammar

# Grammar and systems

LinGO: English resource grammar in linguistic grammars online

Based on HPSG `94, enriched motivated by Sag `97 and with MRS

1,663 abstract lexical types

180 rule types

58 specific phrasal and lexical rules

6,766 lexical entries

79 features for lexical and phrasal types

39 features for lexical semantic predicates(4,562)

# Grammar and systems

PAGE: Platform for Advanced Grammar Engineering

- includes language and interpreter for high-level definition of types feature structures in multiple inheritance hierarchy
- parameterizable chart parser, feature structure viewer, chart viewer

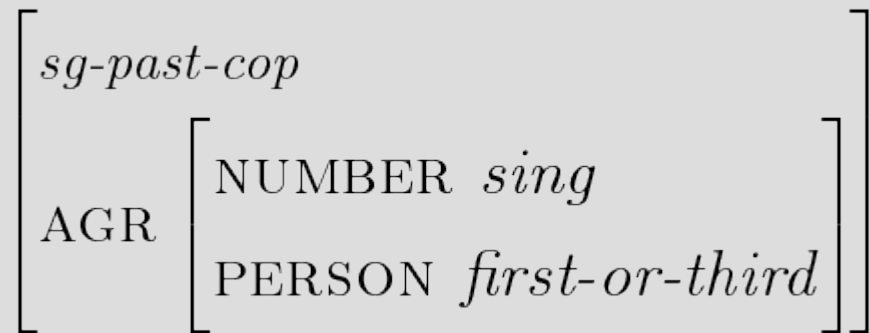
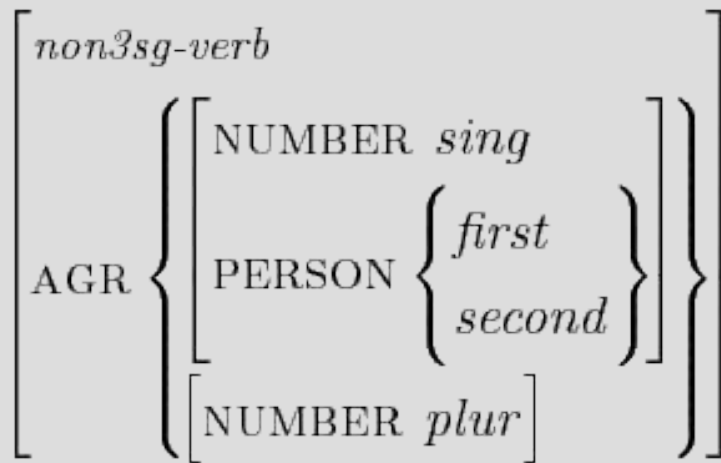
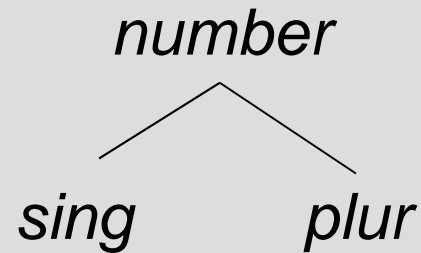
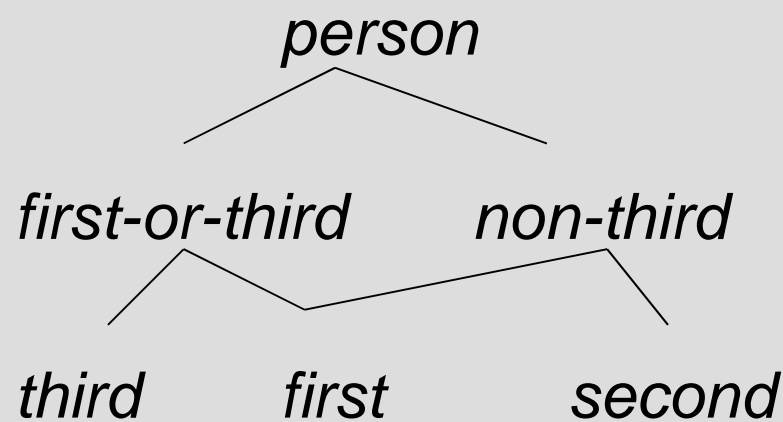
LKB: Linguistic Knowledge building

- chart parser, unifier, tools to view feature structures, parse charts, type hierarchies
- difference to PAGE: each feature is to be introduced on exactly one maximal type (type inference)

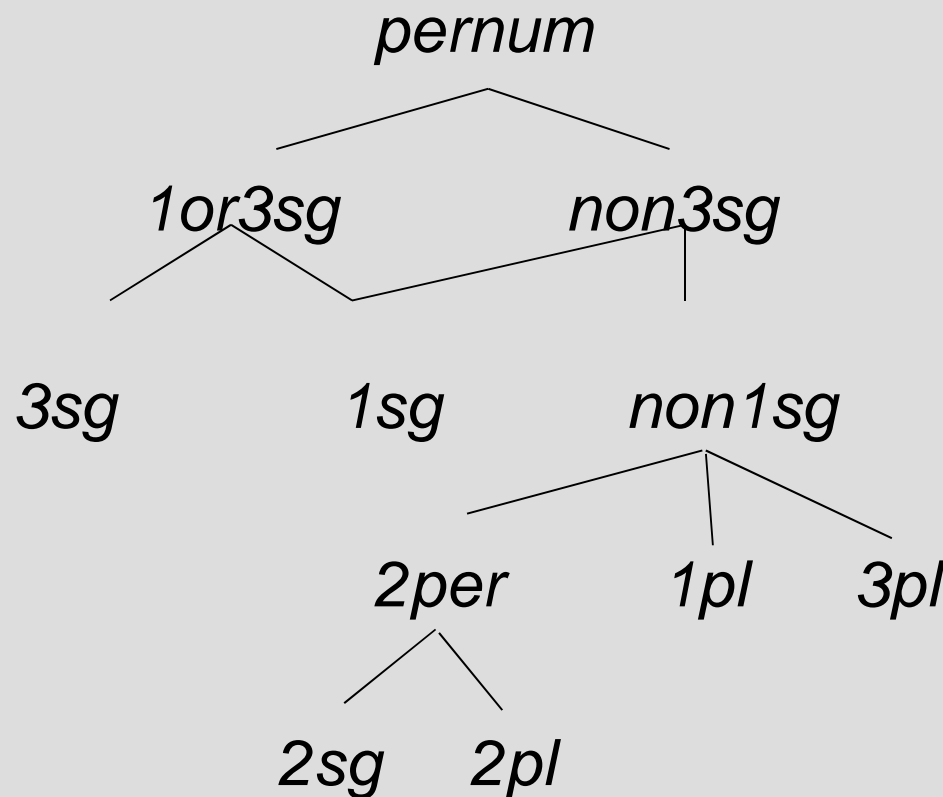
# Disjunctive features

- disjunction is computational expensive
- disjunction is used frequently in describing grammar
- the use of types can supplant the use of disjunctive features

# Disjunctive features : original hierarchy and constraint



# Disjunctive features : modified hierarchy and constraint



$$\left[ \begin{array}{l} non3sg-verb \\ AGR \left[ PERNUM \ non3sg \right] \end{array} \right]$$

- no disjunction in the constraint on non-3-sg-verbs
- the pernum hierarchy only includes relevant types

# disjunctive features - evaluation

- equivalent linguistic description
- gain in efficiency: both time and space

Platform	Test suite	Grammar version	Items	Etasks	Time (s)	Space(kb)
PAGE	TSNLP	with disjunction	4,511	656	3,22	18,016
		no disjunction	4,511	1,006	0,92	5,747
PAGE	`aged`	with disjunction	96	1,684	32,8	68,629
		no disjunction	96	3,885	8,4	24,373

- efficiency gain is partly due to different unifier

# Reducing number of lexical entries

Problem: due to elimination of disjunctive features the number of lexical entries might grow, because of

- optional complements
- different subcategorization frames
- lexical semantic ambiguity

- (4) a Kim asked Sandy a question.  
b Kim asked Sandy.  
c Kim asked a question.  
d Kim asked.

# Reducing number of lexical entries

Two ways to solve the problem:

- move disjunction to lexicon (4 lexical entries)
- encode optionality directly as a property of subcategorized-for elements:

$$\left[ \begin{array}{l} ask \\ \text{SUBJ} \langle \text{NP} [\text{OPT} -] \rangle \\ \text{COMPS} \langle \text{NP} [\text{OPT} +], \text{NP} [\text{OPT} +] \rangle \end{array} \right]$$

# Reducing number of lexical entries ctnd

Two ways to integrate new feature:

- Add rules, that discharge missing optional arguments
  - three new rules: optional specifier, subject and object
  - no changes to any other rules necessary
  - resulting trees may have unnecessary nodes
- introduce new type *optlist*, which unifies either with the empty list or with a list containing only [OPT +] elements
  - replace all occurrences of *elist* by *optlist*
  - no unnecessary nodes

# Reducing number of lexical entries ctnd

Comparing both ways to integrate OPT-feature

Platform	Test suite	Grammar version	Items	Etasks	Time (s)	Space(kb)
LKB	`csl`	no optlist	1,169	706	0.30	1,003
		with optlist	1,172	549	0.23	787

- efficiency gain: 30% less time  
20% less heap space

# Reducing size of feature structures

- total well-typedness leads to large feature structures
- many features with most general value => no gain in information but costly in time and space
- old strategy (Götz `93, Gerdemann&King `94):  
remove those attribute-value pairs after grammar is loaded, reconstruct them after processing
- new strategy: introduce supertypes without attributes appropriate for types like *head*

# Reducing size of feature structures ctnd

- introduce *head\_min* as a supertype of *head* with no features appropriate
- change feature declaration: HEAD has *head\_min* as ist value
- if there is no specific information for this part of structure use *head\_min*

Platform	Test suite	Grammar version	Items	Etasks	Time (s)	Space(kb)
LKB	`csl`	no mintypes	1,348	2,348	2.15	4,620
		with mintypes	1,348	2,396	1.57	3,292

# summary

- Exploiting types can make a grammar much more efficient without making the grammar less elegant in description
- Efficiency is not only subject of parser or generator nor of the grammatical framework