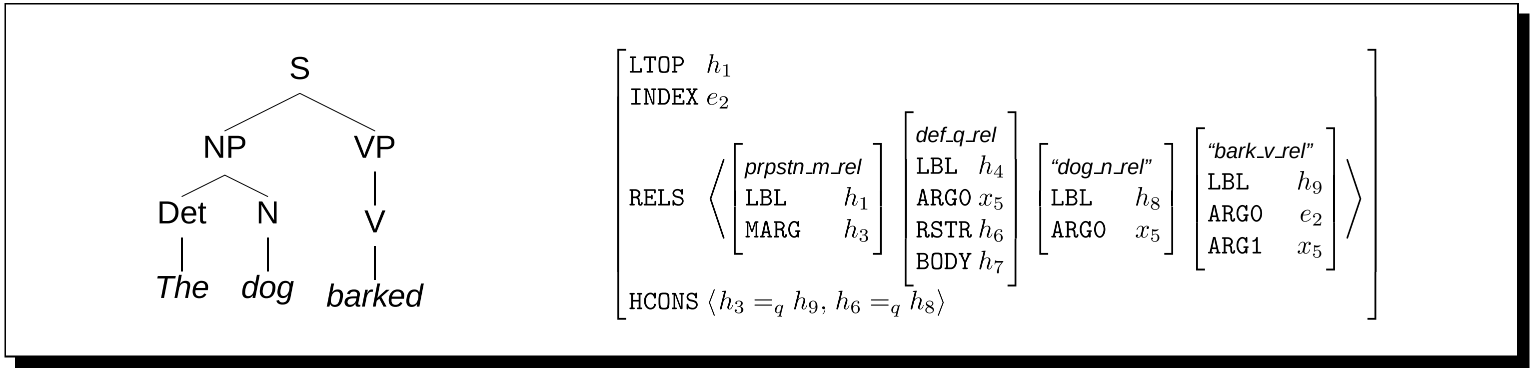
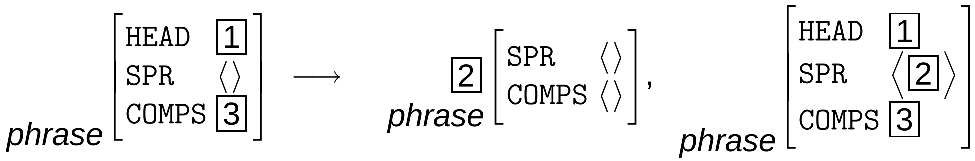


Computational Linguistics (INF2820 — Bits & Pieces)



Stephan Oepen

Universitetet i Oslo

oe@ifi.uio.no

A Highly Ambiguous Example

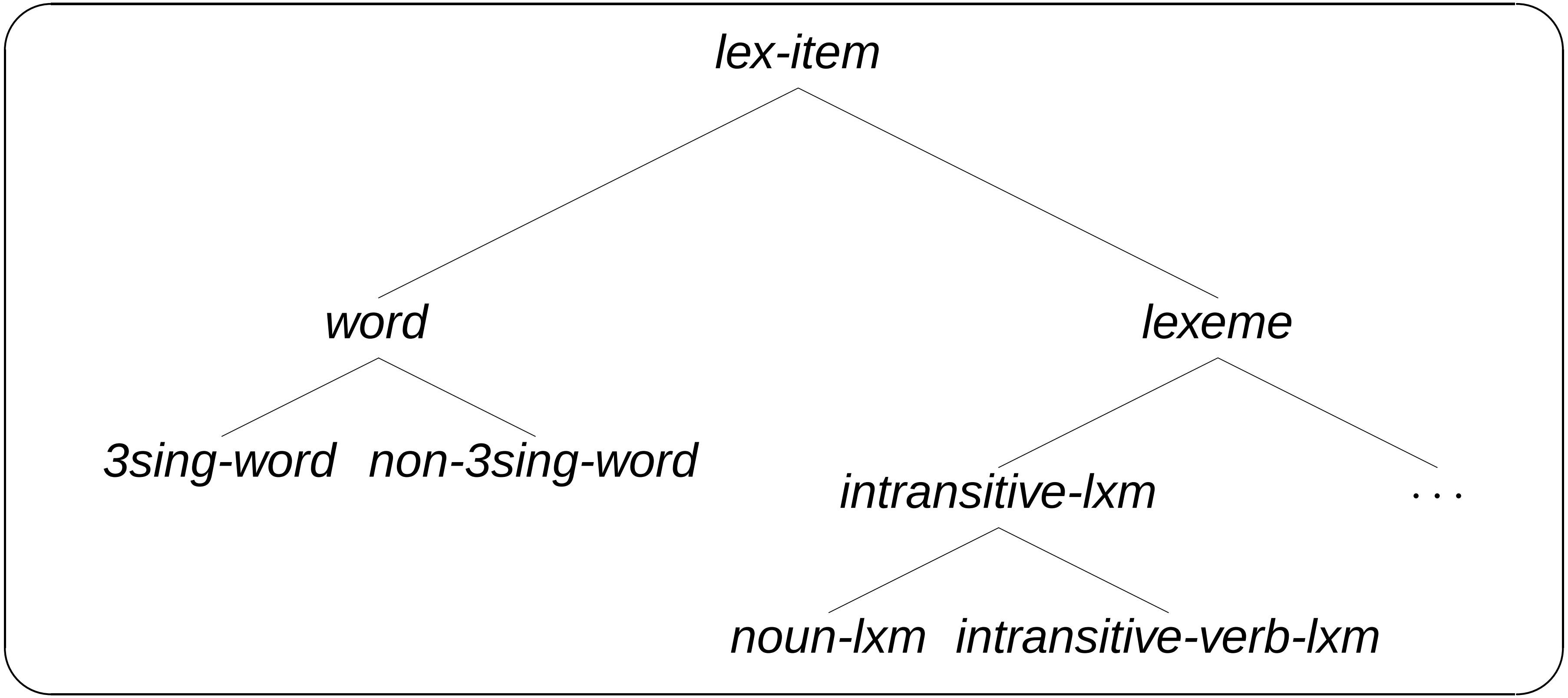
The manager placed his bid on my desk.

Dative Shift: A Productive Process

$$\{\textit{hand}_1, \textit{give}_1, \textit{send}_1, \ldots\} \begin{bmatrix} \texttt{HEAD} & \textit{verb} \\ \texttt{SPR} & \langle \cdots \rangle \\ \texttt{COMPS} & \left\langle {}_{\textit{phrase}}\begin{bmatrix} \texttt{HEAD} & \textit{noun} \\ \texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle \end{bmatrix}, {}_{\textit{phrase}}\begin{bmatrix} \texttt{HEAD} & \textit{noun} \\ \texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\{\textit{hand}_2, \textit{give}_2, \textit{send}_2, \ldots\} \begin{bmatrix} \texttt{HEAD} & \textit{verb} \\ \texttt{SPR} & \langle \cdots \rangle \\ \texttt{COMPS} & \left\langle {}_{\textit{phrase}}\begin{bmatrix} \texttt{HEAD} & \textit{noun} \\ \texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle \end{bmatrix}, {}_{\textit{phrase}}\begin{bmatrix} \texttt{HEAD} & {}_{\textit{prep}}\begin{bmatrix} \texttt{PFORM} & \textit{to} \end{bmatrix} \\ \texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

The Lexeme vs. Word Distinction



- Lexical entries are *uninflected*; cannot enter syntax by themselves;
- inflectional rules ‘make’ *word* from *lexeme*, possibly with ‘null’ suffix.

Orthographemic Variation: Inflectional Rules

```
%(letter-set (!s abcdefghijklmnopqrtuvwxyz))

noun-non-3sing_irule :=
%suffix (!s !ss) (!ss !ssses) (ss sses)
non-3sing-word &
[ HEAD [ AGR non-3sing ],
  ARGS < noun-lxm > ].

noun-3sing_irule :=
3sing-word &
[ ORTH #1,
  ARGS < noun-lxm & [ ORTH #1 ] > ].
```

dog
|
dogs

bus
|
busses

pass
|
passes

Recursion in the Type Hierarchy

- Type hierarchy must be finite *after* type inference; illegal type constraint:

```
*list* := *top* & [ FIRST *top*, REST *list* ].
```

- needs additional provision for empty lists; indirect recursion:

```
*list* := *top*.
*ne-list* := *list* & [ FIRST *top*, REST *list* ].
*null* := *list*.
```

- recursive types allow for *parameterized list types* ('list of X'):

```
*s-list* := *list*.
*s-ne-list* := *ne-list* & *s-list &
               [ FIRST syn-struc, REST *s-list* ].
*s-null* := *null* & *s-list*.
```

Our Grammars: Table of Contents

Type Description Language (TDL)

- `types.tdl` type definitions: hierarchy of grammatical knowledge;
- `lexicon.tdl` instances of (lexical) types plus orthography;
- `rules.tdl` instances of construction types; used by the parser;
- `lrules.tdl` lexical rules, applied before non-lexical rules;
- `irules.tdl` lexical rules that require orthographemic variation;
- `roots.tdl` grammar start symbol(s): 'selection' of final results.

Auxiliary Files (Grammar Configuration for LKB)

- `labels.tdl` TFS templates abbreviating node labels in trees;
- `globals.lsp`, `user-fns.lsp` parameters and interface functions;
- `mrsglobals.lsp` MRS parameters (path to semantics et al.)

LinGO English Resource Grammar

Linguistic Grammars On-Line (`http://lingo.stanford.edu/erg`)

- LinGO English Resource Grammar (Dan Flickinger et al., since 1993);
- general-purpose HPSG; domain-specific lexica (some 32,000 lexemes);
- development using LKB; high-efficiency C[++] parser for applications;
- domain-specific vocabulary addition and tuning → ~85^{+}% coverage;
- average parse times: a few seconds per sentence, for Wikipedia text;

→ exact same resource used simultaneously in many (research) projects.

An Open-Source Repository (`http://www.delph-in.net/`)

- Harmonize theory, formalism, and tools: exchange ling- and software;
- world-wide initiative, now twelve languages under active development.

Review: Context-Free Grammars

- Formally, a *context-free grammar* (CFG) is a quadruple: $\langle C, \Sigma, P, S \rangle$
- C is the set of categories (aka *non-terminals*), e.g. $\{$S, NP, VP, V$\}$;
- Σ is the vocabulary (aka *terminals*), e.g. $\{$Kim, snow, saw, in$\}$;
- P is a set of category rewrite rules (aka *productions*), e.g.

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
NP $\rightarrow$ Kim
NP $\rightarrow$ snow
V $\rightarrow$ saw

- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

The Chomsky Hierarchy of (Formal) Languages

- (Formal) Languages vary in 'degree of structural complexity' exhibited;
- traditionally: a^* (iteration) vs. $a^n b^n$ (nesting) vs. $a^n b^m c^n d^m$ ('cross-serial');
- Chomsky Hierarchy: inclusion classes of formal languages; Type 0 – 3.

0	unrestricted	$\beta_1 \rightarrow \beta_2$	Turing Machine
1	context-sensitive	$\beta_1 \alpha \beta_2 \rightarrow \beta_1 \gamma \beta_2$	linearly-bounded automaton
2	context-free	$\alpha \rightarrow \beta$	push-down automaton
3	regular	$\alpha \rightarrow \delta \mid \alpha\delta$	finite-state automaton

$$\alpha \in C,\ \ \beta_i \in (C \cup \Sigma)^*,\ \ \gamma \in (C \cup \Sigma)^+,\ \ \delta \in \Sigma^+$$

What is the Formal Complexity of Natural Languages?

- Minimally context-free (center self-embedding, e.g. in relative clauses);
- (Culy; Shieber, 1985): *not* context-free (Bambara, Swiss German);
- (Joshi, 1985): extra class of *mildly* context-sensitive languages (TAG).

Adding Semantics to Unification Grammars

- **Logical Form**

  For each sentence admitted by the grammar, we want to produce a meaning representation that is suitable for applying rules of inference.

  This fierce dog chased that angry cat.

  $$\textit{this}(x) \wedge \textit{fierce}(x) \wedge \textit{dog}(x) \wedge \textit{chase}(e,x,y) \\ \wedge \textit{past}(e) \wedge \textit{that}(y) \wedge \textit{angry}(y) \wedge \textit{cat}(y)$$

- **Compositionality**

  The meaning of each phrase is composed of the meanings of its parts.

- **Existing Machinery**

  Unification is the only means for constructing semantics in the grammar.

Appending Lists with Unification

- A *difference list* embeds an open-ended list into a container structure that provides a ‘pointer’ to the end of the ordinary list at the top level:

$$\boxed{A}\ \begin{bmatrix} \textit{*dlist*} \\ \mathrm{LIST}\ \boxed{1}\begin{bmatrix} \textit{*ne-list*} \\ \mathrm{FIRST}\ X \\ \mathrm{REST}\ \boxed{2}\ \textit{*list*} \end{bmatrix} \\ \mathrm{LAST}\ \boxed{2} \end{bmatrix} \qquad \boxed{B}\ \begin{bmatrix} \textit{*dlist*} \\ \mathrm{LIST}\ \boxed{3}\begin{bmatrix} \textit{*ne-list*} \\ \mathrm{FIRST}\ Y \\ \mathrm{REST}\ \boxed{4}\ \textit{*list*} \end{bmatrix} \\ \mathrm{LAST}\ \boxed{4} \end{bmatrix}$$

- Using the `LAST` pointer of difference list $\boxed{A}$ we can append $\boxed{A}$ and $\boxed{B}$ by

  (i) unifying the front of $\boxed{B}$ (i.e. the value of its `LIST` feature) into the tail of $\boxed{A}$ (i.e. the value of its `LAST` feature); and

  (ii) using the tail of $\boxed{B}$ as the new tail for the result of the concatenation.

Notational Conventions

- lists not available as built-in data type; abbreviatory notation in TDL:

```
< a, b > ≡ [ FIRST a, REST [ FIRST b, REST *null* ] ]
```

- underspecified (variable-length) list:

```
< a, ... > ≡ [ FIRST a, REST *list* ]
```

- difference (open-ended) lists; allow concatenation by unification:

```
<! a !> ≡ [ LIST [ FIRST a, REST #tail ], LAST #tail ]
```

- built-in and 'non-linguistic' types pre- and suffixed by asterisk (**top**);
- strings (e.g. *"chased"*) need no declaration; always subtypes of **string**;
- strings cannot have subtypes and are (thus) mutually incompatible.

An Example: Concatenation of Orthography

$$\begin{bmatrix} \text{ORTH} \begin{bmatrix} \text{LIST} & \boxed{1} \\ \text{LAST} & \boxed{3} \end{bmatrix} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{ORTH} \begin{bmatrix} \text{LIST} & \boxed{1} \\ \text{LAST} & \boxed{2} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{ORTH} \begin{bmatrix} \text{LIST} & \boxed{2} \\ \text{LAST} & \boxed{3} \end{bmatrix} \end{bmatrix}$$