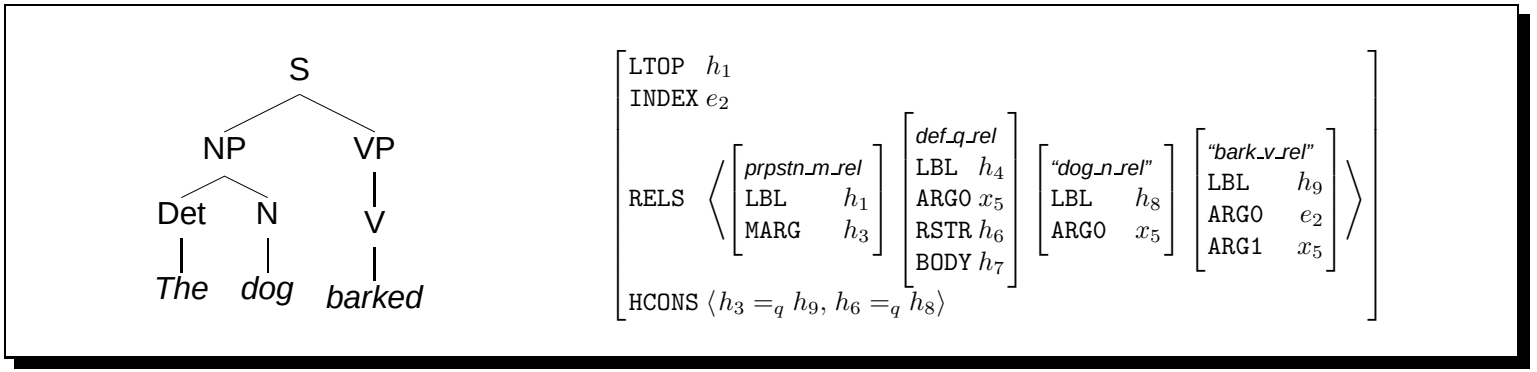


Algorithms for AI and NLP (INF4820 — Unification)

$$\textit{phrase}\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \boxed{3}\end{bmatrix} \longrightarrow \boxed{2}\textit{phrase}\begin{bmatrix}\texttt{SPR} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle\end{bmatrix}, \textit{phrase}\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \langle\boxed{2}\rangle \\ \texttt{COMPS} & \boxed{3}\end{bmatrix}$$

Stephan Oepen and Jan Tore Lønning

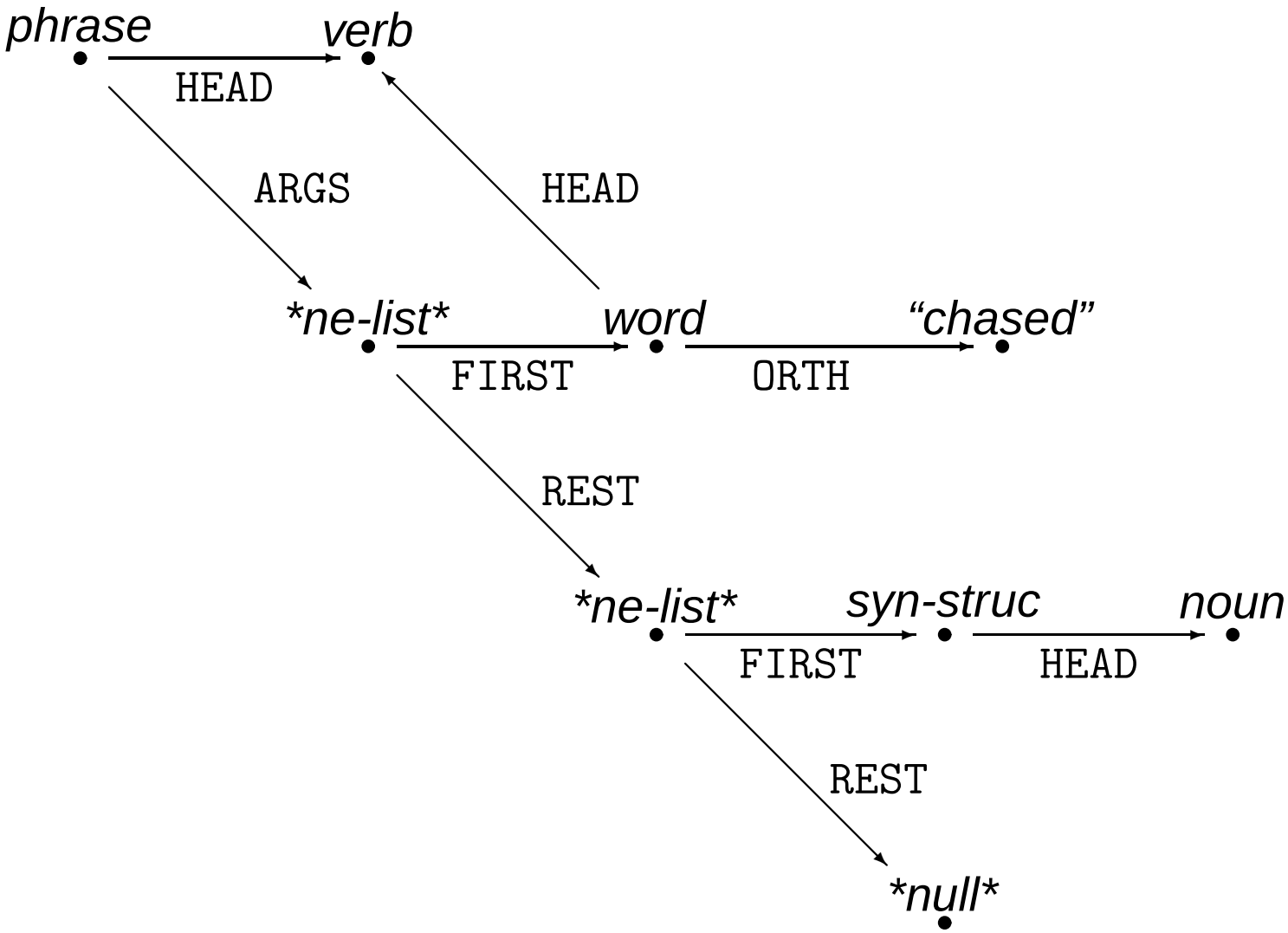
Universitetet i Oslo

{ oe | jtl }@ifi.uio.no

Feature Structure Reentrancy (AVM)

$$\textit{phrase}\left[\begin{array}{ll}\texttt{HEAD}\ \boxed{1}\ \textit{verb} & \\ \texttt{ARGS} & \textit{*ne-list*}\left[\begin{array}{ll}\texttt{FIRST} & \textit{word}\left[\begin{array}{l}\texttt{ORTH}\ \textit{“chased”}\\ \texttt{HEAD}\ \boxed{1}\end{array}\right]\\ \texttt{REST} & \textit{*ne-list*}\left[\begin{array}{ll}\texttt{FIRST} & \textit{syn-struc}\left[\texttt{HEAD}\ \textit{noun}\right]\\ \texttt{REST} & \textit{*null*}\end{array}\right]\end{array}\right]\end{array}\right]$$

Feature Structure Reentrancy (DAG)



Feature Structure Unification & Copying

Basic Notions

- Typed feature structures encoded as *directed acyclic graphs* (DAGs);
- each node bears a *type* and a set of *arcs* (aka features – value pairs);
- feature structure reentrancy (coreference) corresponds to DAG identity;
- unification creates *equivalence classes*, encoded through *forwarding*.

Basic Operations

- *unify()* — make two DAGs equivalent, check and combine all information;

→ at each node, `glb()` types, forward, recurse over and accumulate arcs;

- *copy()* — create *structurally equivalent* copy (preserving reentrancies);

→ at each node, *copy* slot as short-term memory, reset upon completion.

A Practical Example: Unifying Two DAGs

$$foo\begin{bmatrix} \mathtt{A}\ \boxed{1}\ bar\begin{bmatrix}\mathtt{B}\ baz\end{bmatrix} \\ \mathtt{C}\ \boxed{1} \end{bmatrix}$$

$$foo\begin{bmatrix} \mathtt{A}\ bar\begin{bmatrix}\mathtt{B}\ baz\end{bmatrix} \\ \mathtt{C}\ fee\begin{bmatrix}\mathtt{D}\ baz\end{bmatrix} \end{bmatrix}$$

The Costs of Feature Structure Manipulation

Basic Cost Measure

- Visit each DAG node once (node operations 'constant'): *full traversal*;
- *linear* in the number of nodes → upper bound is size of largest DAG.

Naïve Complexity Theory

- Prior to each (destructive) unification, make copies of both input DAGs;
- upon completion of each copy, recursively reset *copy* slot on all nodes.

`restore()`	`copy()`	`unify()`
1	2	5

The unify() vs. copy() Trade-Off

Destructive Unification [Boyer & Moore, 1972]

- Permanently alter both input dags: `setf()` on *forward*, *type*, and *arcs*;

→ *over copying* — two full copies required for only one result structure;

→ *early copying* — majority of unifications fail: many unnecessary copies.

Non-Destructive Unification [Wroblewski, 1987]

- Incrementally build up result DAG during unification, one node at a time;

→ eliminates over copying, reduces early copying more or less effectively.

Quasi-Destructive Unification [Tomabechi, 1991]

- Alter input DAGs in way that is reversible (at small cost): 'generations';

→ copy out result only after unification success, no over or early copying.

Generation Counting

- Protect DAG slots with *generation* counter → 'expiration date' of value;
- access: require valid generation; assignment: set value *and* generation;

→ implemented through interaction of global counter and ADT functionality.

```
(defstruct dag
  forward type arcs xcopy (generation 0))
(defparameter *generation* 1)
(defun dag-copy (dag)
  (when (= (dag-generation dag) *generation*) (dag-xcopy dag)))
(defsetf dag-copy dag-set-copy)
(defun dag-set-copy (dag value)
  (setf (dag-generation dag) *generation*)
  (setf (dag-xcopy dag) value))
```

Unification-Based Grammar: Structured Categories

- All (constituent) categories in the grammar are *typed feature structures*;
- feature structures are recursive, record-like objects: attribute – value sets;
- specific TFS configurations may correspond to 'traditional' categories;
- labels like 'S' or 'NP' are mere abbreviations, not elements of the theory.

$$\textit{word}\begin{bmatrix}\texttt{HEAD} & \textit{noun}\\ \texttt{SPR} & \left\langle\left[\texttt{HEAD}\ \textit{det}\right]\right\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}$$

'N'

$$\textit{phrase}\begin{bmatrix}\texttt{HEAD} & \textit{verb}\\ \texttt{SPR} & \langle\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}$$

'S'

$$\textit{phrase}\begin{bmatrix}\texttt{HEAD} & \textit{verb}\\ \texttt{SPR} & \left\langle\left[\texttt{HEAD}\ \textit{noun}\right]\right\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}$$

'VP'

Interaction of Lexicon and Phrase Structure Schemata

$$\textit{phrase}\begin{bmatrix}\texttt{HEAD} & \boxed{1}\\ \texttt{SPR} & \langle\rangle\\ \texttt{COMPS} & \boxed{3}\end{bmatrix} \longrightarrow \boxed{2}\textit{phrase}\begin{bmatrix}\texttt{SPR} & \langle\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}, \textit{phrase}\begin{bmatrix}\texttt{HEAD} & \boxed{1}\\ \texttt{SPR} & \langle\boxed{2}\rangle\\ \texttt{COMPS} & \boxed{3}\end{bmatrix}$$

$$\begin{bmatrix}\texttt{HEAD} & \textit{noun}\\ \texttt{SPR} & \langle\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}$$

“Kim”

$$\begin{bmatrix}\texttt{HEAD} & \textit{verb}\\ \texttt{SPR} & \left\langle\begin{bmatrix}\texttt{HEAD} & \textit{noun}\\ \texttt{SPR} & \langle\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}\right\rangle\\ \texttt{COMPS} & \langle\rangle\end{bmatrix}$$

“snored”

Unification-Based Parsing

Adaptations to CFG-Based Chart Parser

- Make all elements of Σ, C, and P from the grammar feature structures;
- substitute *unification* and *equivalence test* for category comparison;
- unify category of passive edges with *argument position* of active edges;

$\rightarrow$ *edge* structure LHS is DAG, RHS list of paths to argument positions;

$\rightarrow$ `fundamental-rule()` result of unification is category for new edge;

$\rightarrow$ `pack-edge()` equivalence test: two DAGs contain same information;

- test *spanning* passive edges for compatibility against start symbol S.

`#E[`*id* `: (`*i*`-`*j*`)` *dag* `-->` *edge*$_1$ ... *edge*$_i$ `.` *path*$_{i+1}$ ... *path*$_n$ { *alternates* }* `]`

```
#E[42: (0-8) head-specifier-rule --> 13 . (ARGS REST FIRST)]
```

Unification-Based Parsing—Practical Concerns

Observations

- Typical systems: 90^{+} per cent of parsing time go to DAG manipulation;
- most unifications fail: predict unification failure cheaply, where possible;

$\rightarrow$ *rule filter*: rule feeding relations; *quick check*: most likely failure paths;

- lexicalisation: argument positions in rules may be highly underspecified;

$\rightarrow$ *head-driven* parsing: instantiate RHS bidirectionally, starting from head;

- many unifications fail very early: `copy()` more expensive than `unify()`;

$\rightarrow$ memory is expensive: redo a couple of unfications instead of one copy.

Several orders of magnitude average speed-up by reducing constants

An Example: The LinGO English Resource Grammar

Suggested Background Activities

- Retrieve the model solution for the fourth exercise from the course site;
- compare our solution to your submission; how is ours better (or not)?
- read [Wroblewski, 1987], be sure to understand *over* and *early* copying;
- investigate a call counting scheme for the DAG manipulation routines;
- identify the parts of our earlier parser that require modifications now;
- [Oepen, Flickinger, Tsujii, & Uszkoreit, 2002], Chapters Five and Nine — see whether you can enjoy reading contemporary parsing literature.