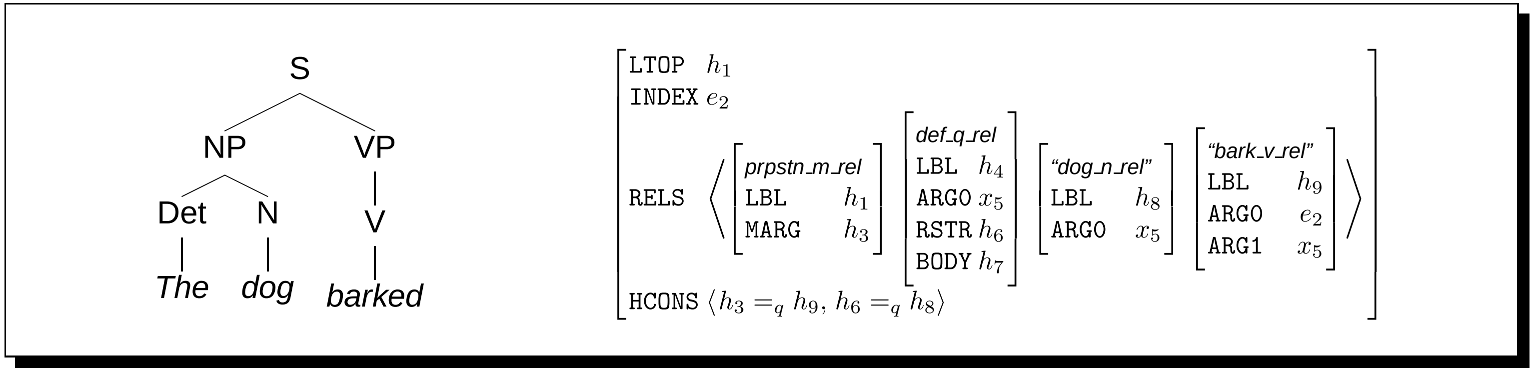

Computational Linguistics (INF2820 — Semantics)

$\{$*this(x)* $\wedge$ *fierce(x)* $\wedge$ *dog(x)* $\wedge$ *bark(e,x)* $\}$

Stephan Oepen

Universitetet i Oslo & CSLI Stanford

oe@ifi.uio.no

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.

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

- **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.

(Elementary) Semantics in Typed Feature Structures

- Semantic content in the SEM attribute of every word and phrase

$$\textit{expression}\begin{bmatrix}\texttt{HEAD} & \textit{pos} \\ \texttt{SPR} & \textit{*list*} \\ \texttt{COMPS} & \textit{*list*} \\ \texttt{SEM} & \textit{semantics}\begin{bmatrix}\texttt{RELS} & \textit{*dlist*}\end{bmatrix}\end{bmatrix}$$

- The value of SEM for a sentence is simply a list of relations in the attribute RELS, with the arguments in those relations 'linked up' appropriately:

$$\begin{bmatrix}\texttt{RELS} & \left\langle \begin{bmatrix}\texttt{PRED} & \textit{"the\_rel"} \\ \texttt{ARG0} & \boxed{1}\textit{entity}\end{bmatrix}, \begin{bmatrix}\texttt{PRED} & \textit{"dog\_rel"} \\ \texttt{ARG0} & \boxed{1}\end{bmatrix}, \begin{bmatrix}\texttt{PRED} & \textit{"bark\_rel"} \\ \texttt{ARG0} & \textit{event} \\ \texttt{ARG1} & \boxed{1}\end{bmatrix} \right\rangle\end{bmatrix}$$

- Semantic relations are introduced by lexical entries, and are appended when grammar rules combine words with other words or phrases.

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:

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

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

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

  (ii) using the tail of $\boxed{\text{B}}$ 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

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

Linking Semantic Arguments

- Each word or phrase also has an `INDEX` attribute in `SEM`
- When heads select a complement or specifier, they constrain its `INDEX` value – an *entity* variable for nouns, an *event* variable for verbs.
- Each lexeme also specifies a `KEY` relation (to allow complex semantics)

$$
\textit{transitive-verb-lxm}\left[\begin{array}{ll}
\mathtt{HEAD} & \textit{verb} \\
\mathtt{SPR.FIRST} & \left[\mathtt{SEM.INDEX}\ \boxed{1}\right] \\
\mathtt{COMPS.FIRST} & \left[\mathtt{SEM.INDEX}\ \boxed{2}\right] \\
\mathtt{SEM} & \left[\begin{array}{ll}
\mathtt{INDEX} & \boxed{0}\ \textit{event} \\
\mathtt{KEY} & \boxed{3}\left[\begin{array}{ll}
\mathtt{PRED} & \textit{*string*} \\
\mathtt{ARG0} & \boxed{0} \\
\mathtt{ARG1} & \boxed{1} \\
\mathtt{ARG2} & \boxed{2}
\end{array}\right] \\
\mathtt{RELS} & \langle\!|\ \boxed{3}\ |\!\rangle
\end{array}\right]
\end{array}\right]
$$

Semantics of Phrases

- Every phrase makes the value of its own `RELS` attribute be the result of appending the `RELS` lists of its daughter(s) (difference list concatenation);
- Every phrase identifies its semantic `INDEX` value with the `INDEX` value of exactly *one* of its daughters (which we will call the *semantic head*);
- As we unify the whole TFS of a complement or specifier with the constraints in the syntactic head, unification takes care of semantic linking.
- Head – modifier structures are analogous: the modifier lexically constrains the `INDEX` of the head daughter it will modify; the rules unify the whole TFS of the head daughter with the `MOD` value in the modifier.

A Linking Example Involving Modification