Computational Linguistics II

— Grammars, Algorithms, Statistics —

Dan Flickinger

Oslo and Stanford Universities

`danf@csli.stanford.edu`

Tore Langholm

Universitetet i Oslo

`torel@ifi.uio.no`

Stephan Oepen

Oslo and Stanford Universities

`oe@csli.stanford.edu`

(1) Unification-Based Grammar

(a) Show the feature structure representation of the following rule as it is used in our LKB grammars:

$$\textit{head-initial}\begin{bmatrix}\texttt{HEAD} & \boxed{1}\\ \texttt{SPR} & \boxed{2}\\ \texttt{COMPS} & \boxed{3}\end{bmatrix} \longrightarrow \textit{expression}\begin{bmatrix}\texttt{HEAD} & \boxed{1}\\ \texttt{SPR} & \boxed{2}\\ \texttt{COMPS} & \begin{bmatrix}\texttt{FIRST} & \boxed{4}\\ \texttt{REST} & \boxed{3}\end{bmatrix}\end{bmatrix}, \quad \boxed{4}\,\textit{phrase}\begin{bmatrix}\,\end{bmatrix}$$

(b) What is the (approximate) name of the above rule in our grammar? Sketch its functionality in a few sentences and provide two examples of types of phrases built using this rule.

(c) Why did we choose to implement rules as single feature structures?

(2) Typed Feature Structures

(a) Draw the following feature structure in DAG notation, i.e. as a directed acyclic graph of labeled nodes and directed arcs:

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

(b) In no more than two sentences, comment on the correspondences between elements of the feature structure and elements of the DAG.

(3) Linguistic Concepts

(a) Name at least two syntactic properties that characterize (syntactic) heads in the formation of phrases. Use one or two examples.

(b) Sketch a constituent tree for the sentence *the fierce dog chased the cat near the aardvark*. On each node, provide information about its general category (using abbreviatory notions like ‘Det’, ‘N’, ‘NP’, ‘VP’, et al.) and for each branch of the tree indicate whether the constituent dominated by it acts as a *head*, *specifier*, *complement*, or *modifier*.

(c) What is the difference between the function of the prepositional phrases (marked by square brackets) in the following two sentences:

(i) *that cat gave the aardvark [to the dogs]*

(ii) *the cat chased the aardvark [near the dogs]*

(4) Parsing and (5) Unification

(a) Which aspect(s) of natural language (grammars) make(s) parsing (using context-free grammars, say) a surprisingly hard problem? How is a naïve parser affected in its performance, and what is the worst-case complexity relative to input string length? Mention at least one example.

(b) How does our generalized chart parser avoid this worst-case complexity, in general and for the example you gave above? Sketch, in a few sentences, the central idea in chart-based parsing.

(a) In a few sentences, recall the notions of *early*- and *over-copying*. Does the generation counting scheme for the *copy* slot of our *dag* structures reduce either one? If so, how? If not, why not?

(6) Common-Lisp

(a) For each of the following lists, make the underlying structure explicit by (i) showing the list in ‘box notation’, and (ii) writing a Common-Lisp expression to construct the list, using exclusively the `cons()` function, the atoms that are elements of the list, and `nil`.

(i) `(1 2 3)`

(ii) `((1) (2 3))`

(b) How many elements are contained in the list returned by the following expression? What will happen when we use the function `length()` to count them?

```
(let ((foo (list 42)))
  (setf (rest foo) foo))
```