Topics in Computational Linguistics — Grammar Engineering —

Dan Flickinger
CSLI Stanford & Saarland University
danf@csli.stanford.edu

Stephan Oepen
Universitetet i Oslo & CSLI Stanford
oe@csli.stanford.edu

http://lingo.stanford.edu/courses/05/ge/
The Linguistic Knowledge Builder (LKB)

General & History
- Specialized grammar engineering environment for TFS grammars;
- main developers: Copestake (original), Carroll, Malouf, and Oepen;
- open-source and binary distributions (Linux, Windows, and Solaris).

Grammar Engineering Fuctionality
- Compiler for typed feature structure grammars $\rightarrow$ wellformedness;
- parser and generator: map from strings to meaning and vice versa;
- visualization: inspect trees, feature structures, intermediate results;
- debugging and tracing: interactive unification, ‘stepping’, et al.
The Type Hierarchy: Fundamentals

- Types ‘represent’ groups of entities with similar properties (‘classes’);
- types ordered by specificity: subtypes inherit properties of (all) parents;
- type hierarchy determines which types are compatible (and which not).
Properties of (Our) Type Hierarchies

- **Unique Top**: a single hierarchy of all types with a unique top node;
- **No Cycles**: no path through the hierarchy from one type to itself;
- **Unique Greatest Lower Bounds**: Any two types in the hierarchy are either (a) incompatible (i.e. share no descendants) or (b) have a unique most general (‘highest’) descendant (called their greatest lower bound);
- **Closed World**: all types that exist have a known position in hierarchy;
- **Compatibility**: type compatibility in the hierarchy determines feature structure unifiability: two types unify to their greatest lower bound.
Multiple Inheritance

- *flyer* and *swimmer* have no common descendants: they are incompatible;
- *flyer* and *bee* stand in hierarchical relationship: they unify to subtype;
- *flyer* and *invertebrate* have a unique greatest common descendant.

```
*top*
    `animal`
    `flyer`
    `swimmer`
    `invertebrate`
    `vertebrate`
        `bee`
        `fish`
            `cod`
            `guppy`
```
An Invalid Type Hierarchy

- *swimmer* and *vertebrate* have two joint descendants: *fish* and *whale*;
- *fish* and *whale* are incomparable in the hierarchy: glb condition violated.
Fixing the Type Hierarchy

- LKB system introduces glb types as required: ‘swimmer-vertebrate’.
Properties of Typed Feature Structures

- **Finiteness**  a typed feature structure has a finite number of nodes;

- **Unique Root and Connectedness**  a typed feature structure has a unique root node; apart from the root, all nodes have at least one parent;

- **No Cycles**  no node has an arc that points back to the root node or to another node that intervenes between the node itself and the root;

- **Unique Features**  any node can have any (finite) number of outgoing arcs, but the arc labels (i.e. features) must be unique within each node;

- **Typing**  each node has single type which is defined in the hierarchy.
Typed Feature Structure Example (as AVM)
Typed Feature Structure Example (as Graph)

phrase → verb

HEAD

ARGS

*ne-list*

FIRST

REST

expression → noun

FIRST

HEAD

REST

*null*
Typed Feature Structure Example (in TDL)

\[
vp := \text{phrase} \& \\
\quad [ \text{HEAD verb}, \\
\quad \text{ARGS} *\text{ne-list}\* \& \\
\quad \quad [ \text{FIRST word} \& \\
\quad \quad \quad [ \text{ORTH} "\text{chased}" , \\
\quad \quad \quad \text{HEAD verb} ], \\
\quad \quad \text{REST} *\text{ne-list}\* \& \\
\quad \quad \quad [ \text{FIRST expression} \& \\
\quad \quad \quad \quad [ \text{HEAD noun} ], \\
\quad \quad \quad \text{REST} *\text{null}\* \] ] ] .
\]
Reentrancy in a Typed Feature Structure (Graph)

phrase

verb

*ne-list*

word

"chased"

expression

noun

*ne-list*

null*
Reentrancy in a Typed Feature Structure (AVM)
Reentrancy in a Typed Feature Structure (TDL)

\[
\begin{align*}
vp & := \text{phrase} \& \\
& \quad [ \text{HEAD } \#\text{head} \& \text{verb}, \\
& \quad \text{ARGS } *\text{ne-list}* \& \\
& \quad \quad [ \text{FIRST word} \& \\
& \quad \quad \quad [ \text{ORTH } "\text{chased}" , \\
& \quad \quad \quad \quad \text{HEAD } \#\text{head} ], \\
& \quad \quad \text{REST } *\text{ne-list}* \& \\
& \quad \quad \quad \quad [ \text{FIRST expression} \& \\
& \quad \quad \quad \quad \quad [ \text{HEAD } \text{noun} ], \\
& \quad \quad \quad \quad \text{REST } *\text{null}* ]] ]
\end{align*}
\]
Typed Feature Structure Subsumption

- Typed feature structures can be partially ordered by information content;
- a more general structure is said to *subsume* a more specific one;
- $*\text{top}*[]$ is the most general feature structure (while $\bot$ is inconsistent);
- $\sqsubseteq$ (‘square subset or equal’) conventionally used to depict subsumption.

Feature structure $F$ subsumes feature structure $G$ ($F \sqsubseteq G$) iff: (1) if path $p$ is defined in $F$ then $p$ is also defined in $G$ and the type of the value of $p$ in $F$ is a supertype or equal to the type of the value of $p$ in $G$, and (2) all paths that are reentrant in $F$ are also reentrant in $G$. 
Feature structure $F$ subsumes feature structure $G$ ($F \sqsubseteq G$) iff: (1) if path $p$ is defined in $F$ then $p$ is also defined in $G$ and the type of the value of $p$ in $F$ is a supertype or equal to the type of the value of $p$ in $G$, and (2) all paths that are reentrant in $F$ are also reentrant in $G$. 

Signature

$$
\begin{array}{c}
    a & \text{FOO} & x \\
    & \text{BAR} & y \\
\end{array}
$$

$$
\begin{array}{c}
    a & \text{BAR} & x \\
\end{array}
$$

$$
\begin{array}{c}
    b & \text{BAZ} & y \\
\end{array}
$$
Typed Feature Structure Unification

- Decide whether two typed feature structures are mutually compatible;
- determine combination of two TFSs to give the most general feature structure which retains all information which they individually contain;
- if there is no such feature structure, unification fails (depicted as \( \bot \));
- unification *monotonically* combines information from both ‘input’ TFSs;
- relation to subsumption: the unification of two structures \( F \) and \( G \) is the most general TFS which is subsumed by both \( F \) and \( G \) (if it exists).
- \( \sqcap \) (‘square set intersection’) conventionally used to depict unification.
Typed Feature Structure Unification: Examples

\[
\begin{align*}
\text{TFS}_1: & \quad \begin{bmatrix} \text{FOO} & x \\text{BAR} & x \end{bmatrix} \\
\text{TFS}_2: & \quad \begin{bmatrix} \text{FOO} & x \\text{BAR} & y \end{bmatrix} \\
\text{TFS}_3: & \quad \begin{bmatrix} \text{FOO} & y \\text{BAR} & x \\
& \text{BAZ} & x \end{bmatrix} \\
\text{TFS}_4: & \quad \begin{bmatrix} \text{FOO} & 1 & x \ \text{BAR} & 1 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{Signature} & \quad \begin{bmatrix} a \text{FOO} & x \ 
\text{BAR} & \end{bmatrix} \\
& \quad \begin{bmatrix} a \text{BAR} & x \ 
\text{BAZ} & y \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{TFS}_1 \sqcap \text{TFS}_2 & \equiv \text{TFS}_2 \\
\text{TFS}_1 \sqcap \text{TFS}_3 & \equiv \text{TFS}_3 \\
\text{TFS}_3 \sqcap \text{TFS}_4 & \equiv \begin{bmatrix} \text{FOO} & 1 & y \\
\text{BAR} & 1 \\
& \text{BAZ} & x \end{bmatrix}
\end{align*}
\]
Type Constraints and Appropriate Features

- Well-formed TFSs satisfy all *type constraints* from the type hierarchy;
- type constraints are typed feature structures associated with a type;
- the top-level features of a type constraint are *appropriate features*;
- type constraints express generalizations over a ‘class’ (set) of objects.

<table>
<thead>
<tr>
<th>type</th>
<th>constraint</th>
<th>appropriate features</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ne-list</em></td>
<td>[FIRST <em>top</em>] REST <em>list</em></td>
<td>FIRST and REST</td>
</tr>
</tbody>
</table>

FIRST and REST
Type Inference: Making a TFS Well-Formed

- Apply all type constraints to convert a TFS into a well-formed TFS;
- determine most general well-formed TFS subsumed by the input TFS;
- specialize all types so that all features are appropriate:

\[
\begin{array}{c}
\text{top} \\
\text{ARGS *list*} \\
\end{array}
\xrightarrow{\text{phrase}}
\begin{array}{c}
\text{HEAD pos} \\
\text{ARGS *list*} \\
\end{array}
\]

- expand all nodes with the type constraint of the type of that node:

\[
\begin{array}{c}
\text{phrase} \\
\text{ARGS *list*} \\
\end{array}
\xrightarrow{\text{phrase}}
\begin{array}{c}
\text{HEAD pos} \\
\text{ARGS *list*} \\
\text{COMPS *list*} \\
\end{array}
\]
More Interesting Well-Formed Unification

Type Constraints Associated to Earlier *animal* Hierarchy

\[
\text{swimmer} \rightarrow \begin{cases} \text{swimmer} & \text{FINS bool} \\ \text{mammal} & \text{FRIENDLY bool} \end{cases}
\]

\[
\text{whale} \rightarrow \begin{cases} \text{mammal} & \text{BALEEN bool} \\ \text{FINS} & \text{true} \\ \text{FRIENDLY bool} \end{cases}
\]

\[
\begin{align*}
\text{mammal}[\text{FRIENDLY true}] \sqcap \text{swimmer}[\text{FINS bool}] &= \begin{cases} \text{mammal} & \text{BALEEN bool} \\ \text{FINS} & \text{true} \\ \text{FRIENDLY true} \end{cases} \\
\text{mammal}[\text{FRIENDLY true}] \sqcap \text{swimmer}[\text{FINS false}] &= \bot
\end{align*}
\]
Recursion in the Type Hierarchy

- Type hierarchy must be finite after type inference; illegal type constraint:
  \[ *\text{list}^{*} := *\text{top}^{*} & \left[ \text{FIRST} *\text{top}^{*}, \text{REST} *\text{list}^{*} \right]. \]

- needs additional provision for empty lists; indirect recursion:
  \[ *\text{list}^{*} := *\text{top}^{*}. \]
  \[ *\text{ne-list}^{*} := *\text{list}^{*} & \left[ \text{FIRST} *\text{top}^{*}, \text{REST} *\text{list}^{*} \right]. \]
  \[ *\text{null}^{*} := *\text{list}^{*}. \]

- recursive types allow for parameterized list types (‘list of X’):
  \[ *\text{s-list}^{*} := *\text{list}^{*}. \]
  \[ *\text{s-ne-list}^{*} := *\text{ne-list}^{*} & *\text{s-list}^{*} & \left[ \text{FIRST} \text{expression}, \text{REST} *\text{s-list}^{*} \right]. \]
  \[ *\text{s-null}^{*} := *\text{null}^{*} & *\text{s-list}^{*}. \]
Notational Conventions

- lists not available as built-in data type; abbreviatory notation in TDL:
  \[< a, b > \equiv [ \text{FIRST} a, \text{REST} [ \text{FIRST} b, \text{REST} *\text{null}* ] ]\]

- underspecified (variable-length) list:
  \[< a \ldots > \equiv [ \text{FIRST} a, \text{REST} *\text{list}]* ]\]

- difference (open-ended) lists; allow concatenation by unification:
  \[<! a !> \equiv [ \text{LIST} [ \text{FIRST} a, \text{REST} \text{#tail}], \text{LAST} \text{#tail} ]\]

- 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.
Recognizing the Language of a Grammar

S → NP VP
VP → V NP
VP → VP PP
NP → NP PP
PP → P NP
NP → kim | snow | oslo
V → snores | adores
P → in

All Complete Derivations
• are rooted in the start symbol S;
• label internal nodes with categories ∈ C, leafs with words ∈ Σ;
• instantiate a grammar rule ∈ P at each local subtree of depth one.
Structured Categories in a Unification Grammar

- All (constituent) categories in the grammar are typed feature structures;
- specific TFS configurations may correspond to ‘traditional’ categories;
- labels like ‘S’ or ‘NP’ are mere abbreviations, not elements of the theory.

```
word
  HEAD noun
  SPR ⟨⟩
  COMPS ⟨⟩

phrase
  HEAD verb
  SPR ⟨⟩
  COMPS ⟨⟩

phrase
  HEAD verb
  SPR ⟨⟩
  COMPS ⟨⟩

‘N’  ‘S’  ‘VP’
‘lexical’ ‘maximal’  ‘intermediate’
```
The Format of Grammar Rules in the LKB

```
mother

daughter1

daughter2

mother
```

ABabcdfghiejkl

stanford

stanford | 6 Jan 05 (oe@csl.i.stanford.edu)

Computational Linguistics: Grammar Engineering (36)
Our Grammars: Table of Contents

<table>
<thead>
<tr>
<th>Type Description Language (TDL)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• types.tdl  type definitions: hierarchy of grammatical knowledge;</td>
<td></td>
</tr>
<tr>
<td>• lexicon.tdl instances of (lexical) types plus orthography;</td>
<td></td>
</tr>
<tr>
<td>• rules.tdl  instances of construction types; used by the parser;</td>
<td></td>
</tr>
<tr>
<td>• lrules.tdl lexical rules, applied before non-lexical rules;</td>
<td></td>
</tr>
<tr>
<td>• irules.tdl lexical rules that require orthographemic variation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auxiliary Files (Grammar Configuration for LKB)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• globals.lsp. Parameter settings (e.g. path to orthography et al.);</td>
<td></td>
</tr>
<tr>
<td>• user-fns.lsp (small number) of LKB interface functions;</td>
<td></td>
</tr>
<tr>
<td>• mrsglobals.lsp MRS parameters (path to semantics et al.)</td>
<td></td>
</tr>
</tbody>
</table>