LSA.343
Precision Grammar Implementation for Linguistic Hypothesis Testing

Emily M. Bender
University of Washington
ebender@u.washington.edu

Dan Flickinger
CSLI Stanford
danf@csli.stanford.edu

Stephan Oepen
Universitet i Oslo & CSLI Stanford
oe@csli.stanford.edu
Regularities in the lexicon

- Distinct words with the same syntactic distribution
  Formal mechanism: Hierarchy of typed feature structures
  e.g. *det-word*, *noun-word-3sing*

- Morphologically related words with distinct syntactic distribution
  Formal mechanism: Lexical rules
  e.g. *reach/reachable*, *open/unopened*, *give(NP,NP)/give(NP,PP)*
Why not do everything with underspecified types?

- No mechanism for forcing an instance to be a most specific subtype

\[
\text{verb-two-complements} \\
\text{\textbackslash} \\
\text{\textbackslash} \\
\text{verb-NP-NP} \quad \text{verb-NP-PP}
\]

give-v1 := verb-two-complements & [ORTH ‘‘give’’].

- Some derivational morphology is recursive

\textit{No anti-anti-ballistic missiles have been built yet.}

\textit{We should add another module to our pre-pre-processing engine.}
Why not underspecified types? (cont.)

- Differences in syntactic properties can be radical
  
  *The dogs were chasing the cats.*
  
  *The chasing of the cats was encouraged.*

- Morphology and syntax can interact
  
  *John pre- and post-dates his checks in desperation.*
Lexical rules

- Function: Derive a new lexical entry from an existing one
- Implementation: Unary rules (redundancy rules or productive rules)
- Types:
  - **Inflectional rules**: Create a word with specific values for properties that are often morphologically marked (e.g. person, number, case, tense, aspect)
  - **Derivational rules**: Create a new word with distinct syntactic properties (e.g. active/passive, dative alternation)
Lexemes, words, and phrases

- We treat lexical entries in our grammar as instances of the type \textit{lexeme}, which are uninflected.

  \begin{verbatim}
  dog-n-lexeme
  [ ORTH ‘‘dog’’,
    AGR pernum ].
  \end{verbatim}

- Inflectional rules take an instance of the type \textit{lexeme} as input, and produce a feature structure of type \textit{word}.

  \begin{verbatim}
  word
  [ ORTH ‘‘dogs’’,
    AGR non-3sing ].
  \end{verbatim}
Lexemes, words, and phrases (cont.)

- Derivational rules can be lexeme-to-lexeme or word-to-word rules.
  - Agentive: \textit{sing} \implies \textit{singer} \implies \textit{singers}
  - Passive: \textit{chase} \implies \textit{chased (the cat)} \implies \textit{(was) chased}

- Phrase structure rules require their daughters to be either of type \textit{word}
  or \textit{phrase}, ensuring that words are all inflected.
Lexical rules as unary rules in the LKB

- The **input** to a rule is encoded as the value of the ARGS attribute, using a typed feature structure to constrain the range of lexemes the rule can apply to.

- The **output** of a rule is encoded as the feature structure of the mother, defining the attributes of the sign being built.

```
lexeme-or-word &
[ HEAD ..., 
  SPR ..., 
  COMPS ..., 
  ARGS < lexeme-or-word & 
    [ HEAD ..., 
      SPR ..., 
      COMPS ... ] > ]
```
The dog **chases** that cat.

That cat was **chased**.

**Sample lexical rule: Passive (simplified)**
Orthographemic Variation: ‘Inflectional’ Rules

\%(letter-set (!s abcdefghijklmnopqrstuvwxyz))

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

noun-3sing_irule :=
word &
[ ORTH #1,
  HEAD [ AGR 3sing ],
  ARGS < noun-lxm & [ ORTH #1 ] > ].
Recursion in the Type Hierarchy

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

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

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