Dependency Parsing and Feature-based Parsing

Ling 571 — Deep Processing Techniques for NLP
October 21, 2019
Shane Steinert-Threlkeld
Announcements

- Thanks for the feedback!
- HW3: mean 92
- Handling ungrammaticality:
  - Need graceful treatment of the case when $S$ / start symbol is not in the $[0, n]$ cell of the CKY table
- Reference code available (in hw3/reference/)
  - example_cky.py in hw4 directory is a symlink to that reference code
HW #4 Notes
HW4 Notes

● If your improvement is along a dimension not measured by evalb (e.g. runtime):
  ● Still run evalb on both old and improved code and report both results
  ● NB: improved runtime cannot occur at “drastic” reduction in accuracy
  ● Write code to measure your performance, and report before/after results in the readme
HW #4: OOV Handling

- As we discussed previously, you will find OOV tokens
- Sometimes this as as simple as case-sensitivity:
Sentence #23: “Arriving before four p.m.”

“arriving” is in our grammar, but not “Arriving”
OOV: Case Sensitivity

Sentence #23: “Arriving before four p.m.”
HW #4: OOV Handling

- Propose some number of N most likely tags at runtime...
**OOV: Propose POS Tags**

"Show me Ground transportation in Denver during weekdays." — No "during"!

| 5 | FRAG_NP_PRIME → 2FRAG_NP_PRIME 4 PP 6[-21.810] |
|   | FRAG_NP → 2FRAG_NP_PRIME 4 PP 6[-20.858] |
|   | NP_PRIME → 3 NN 4 PP 6[-16.296] |
|   | PRIME → 3 NN 4 PP 6[-15.949] |
|   | IN → "in" [-2.4018] |
|   | PP → 4 IN 5 NP_NNP 6[-7.505] |
|   | FRAG_PP → 4 IN 5NP_NNP 6 [-6.828] |
| 6 | NNP → "Denver" [-4.4002] |
|   | NP_NNP → "Denver" [-3.3280] |
| 7 | NNS → "weekdays" [-5.5759] |
|   | NP_NNS → "weekdays" [-3.7257] |
| 8 | TOP → 7NP_NNS 8PUNC 9[-11.001] |
|   | PUNC → "." [-0.3396] |
## OOV: Propose POS Tags

"Show me Ground transportation in Denver during weekdays." — No “during”!

| NP_PRIME → … | TOP → 2FRAG_NP 8 PUNC 9[-34.939] |
| PRIME → 3 NN 4PP 7 [-17.145] | TOP → 3NP 8PUNC 9[-29.022] |
| QP → 3 PRIME 6CD 7 [-15.930] | TOP → 3NP 8PUNC 9[-28.877] |

| PP → … | TOP → 4PP 8PUNC 9[-24.540] |
| FRAG_PP → … | TOP → 4FRAG_PP 8 PUNC 9[-23.716] |
| PP → 4 IN 5 NP 7[-8.701] | NP → 3 PRIME 7NNS 8 [-26.542] |
| FRAG_PP → 4 IN 5NP 7 [-7.878] | NP → 3 QP 7 NNS 8 [-26.398] |

| NNP → "Denver" [-4.4002] | TOP → 5NP 8PUNC 9[-19.809] |
| NP_NNP → "Denver" [-3.3280] | TOP → 5NP 8PUNC 9[-17.905] |
| NP_PRIME → 5NNP 6 NNP 7[-6.110] | NP → 5 NP 7 NNS 8 [-17.330] |
| NP → 5 NNP 6NNP 7 [-5.070] | NP → 5NP_PRIME 7 NNS 8 [-15.426] |

- **NNP**: "during" [1.0000]
- **NN**: "during" [1.0000]
- **NP_NNP**: "during" [1.0000]
- **VB**: "during" [1.0000]
- **CD**: "during" [1.0000]

| VP → 6 VB 7NP_NNS 8[-8.922] | TOP → 6VP 8PUNC 9[-11.410] |
| S_VP → 6 VB 7NP_NNS 8[-6.611] | TOP → 6S_VP 8PUNC 9[-9.176] |

| NNS → "weekdays" [-5.5759] | TOP → 7NP_NNS 8 PUNC 9[-11.001] |
| NP_NNS → "weekdays" [-3.7257] | PUNC → "." [-0.3396] |
"Show me Ground transportation in Denver during weekdays." — No “during”!

Parse result:
“Show me Ground transportation in Denver during weekdays.” — No “during”!

Gold parse:

```
TOP
  S_VP
    S_VP_PRIME
      VB | NP_PRP
      NP
        NP_PRIME
          NP | PP
            NP | IN | NP_NNS
      PP
```

"Show me Ground transportation in Denver during weekdays."
Problems with this approach?
Handling OOV

● **Option #1:**
  ● Choose subset of training data vocab to be hidden
  ● Hidden words replaced by `<UNK>`
  ● Run induction as usual, but some words are now ‘<UNK>’

● **Option #2:**
  ● Implicit vocab creation:
    ● Replace all words occurring less than $n$ times with `<UNK>`
    ● Fix size of $V$ (e.g. 50,000), anything not among $|V|$ most frequent is `<UNK>`

● (See J&M 2nd ed 4.3.2 — 3rd ed, 3.3.1)
Problems with These Approaches?

- **Option #1**
  - May sample “closed-class” words
  - Closed-class words are disproportionately more common
  - ∴ Approximation will be worse the more data there is, *because Zipf*

- **Option #2**
  - **Con**: Requires a lot more data
  - **Pros**: Samples from all word classes
    - Will only count closed-class words once
Today

● Dependency Parsing
  ● Transition-based Parsing

● Feature-based Parsing
  ● Motivation
  ● Features
  ● Unification
### Dependency Parse Example:

*They hid the letter on the shelf*

#### Argument Dependencies

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>nominal subject</td>
</tr>
<tr>
<td>csubj</td>
<td>clausal subject</td>
</tr>
<tr>
<td>dobj</td>
<td>direct object</td>
</tr>
<tr>
<td>iobj</td>
<td>indirect object</td>
</tr>
<tr>
<td>pobj</td>
<td>object of preposition</td>
</tr>
</tbody>
</table>

#### Modifier Dependencies

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmod</td>
<td>temporal modifier</td>
</tr>
<tr>
<td>appos</td>
<td>appositional modifier</td>
</tr>
<tr>
<td>det</td>
<td>determiner</td>
</tr>
<tr>
<td>prep</td>
<td>prepositional modifier</td>
</tr>
</tbody>
</table>

![Dependency Parse Diagram]

They

hid

nsubj
dobj

They

letter

det

on

the

det

shelf

det

the
Transition-Based Parsing

● Parsing defined in terms of sequence of transitions

● Alternative methods for learning/decoding
  ● Most common model: Greedy classification-based approach
  ● Very efficient: $O(n)$

● Best-known implementations:
  ● Nivre’s MALTParser
Transition-Based Parsing

- A transition-based system for dependency parsing is:
  - A set of configurations $C$
  - A set of transitions between configurations
  - A transition function between configurations
  - An initialization function (for $C_0$)
  - A set of terminal configurations (“end states”)
Configurations

- A configuration for a sentence $x$ is the triple $(\Sigma, B, A)$:
  - $\Sigma$ is a stack with elements corresponding to the nodes (words + ROOT) in $x$
  - $B$ (aka the buffer) is a list of nodes in $x$
  - $A$ is the set of dependency arcs in the analysis so far,
    - $(w_i, L, w_j)$, where $w_x$ is a node in $x$ and $L$ is a dependency label
Transitions

- Transitions convert one configuration to another
  - $C_i = t(C_{i-1})$, where $t$ is the transition
- Dependency graph for a sent:
  - The set of arcs resulting from a sequence of transitions
  - The parse of the sentence is that resulting from the initial state through the sequence of transitions to a legal terminal state
Dependencies → Transitions

- To parse a sentence, we need the sequence of transitions that derives it
- How can we determine sequence of transitions, given a parse?
- This is defining our *oracle* function:
  - How to take a parse and translate it into a series of transitions
Dependencies → Transitions

- Many different oracles:
  - Nivre’s arc-standard
  - Nivre’s arc-eager
  - Non-projectivity with Attardi’s
  - ...

- Generally:
  - Use oracle to identify gold transitions
  - Train classifier to predict best transition in new config
Nivre’s Arc-Standard Oracle

- Words: $w_1, \ldots, w_n$
  - $w_0 = \text{ROOT}$

- Initialization:
  - Stack = $[w_0]$; Buffer = $[w_1, \ldots, w_n]$; Arcs = $\emptyset$

- Termination:
  - Stack = $\sigma$; Buffer = $[\ ]$; Arcs = $A$
  - for any $\sigma$ and $A$
Nivre’s Arc-Standard Oracle

- Transitions are one of three:
  - Shift
  - Left-Arc
  - Right-Arc
Transitions: Shift

- *Shift* first element of buffer to top of stack.
- \([i][j,k,n,...][] \rightarrow [i,j][k,n,...][]\)
Transitions: Shift

- *Shift* first element of buffer to top of stack.
- \([i][j,k,n,\ldots][] \rightarrow [i,j][k,n,\ldots][]\)
Transitions: Left-Arc

- Add arc from element at top of stack to second element on stack with dependency label l
- Pop second element from stack.
- \([i,j] [k,n,...] A \rightarrow [j] [k,n,...] A \cup [(j,l,i)]\)
Transitions: Left-Arc

- Add arc from element at top of stack to second element on stack with dependency label \( l \)
- Pop second element from stack.
- \([i,j] [k,n,…] \rightarrow [j] [k,n,…] A \cup [(j,l,i)]\)

Stack  Buffer  Arcs
Transitions: Right-Arc

- Add arc from second element on stack to top element on stack with dependency label $l$
- Pop top element from stack.
- $[i,j] [k,n,...] A \rightarrow [j] [k,n,...] A \cup [(i,l,j)]$

Stack

Buffer

Arcs
Transitions: Left-Arc

- Add arc from second element on stack to top element on stack with dependency label $l$
- Pop top element from stack.
- $[i,j] [k,n,...]$ $\rightarrow$ $[j] [k,n,...]$ $\cup [(i,l,j)]$
Training Process

- Each step of the algorithm is a decision point between the three states
- We want to train a model to decide between the three options at each step
  - (Reduce to a classification problem)
- We start with:
  - A treebank
  - An oracle process for guiding the transitions
  - A discriminative learner to relate the transition to features of the current configuration
Training Process, Formally:

\((\Sigma, B, A)\)

1) \(c \leftarrow c_0(S)\)
2) while \(c\) is not terminal
3) \(t \leftarrow o(c)\) # Choose the (o)ptimal transition for the config \(c\)
4) \(c \leftarrow t(c)\) # Move to the next configuration
5) return \(G_c\)
Testing Process, Formally:

\((\Sigma, B, A)\)

1) \(c \leftarrow c_0(S)\)
2) while \(c\) is not terminal
3) \(t \leftarrow \lambda_c(c)\) # Choose the transition given model parameters at \(c\)
4) \(c \leftarrow t(c)\) # Move to the next configuration
5) return \(G_c\)
Representing Configurations with Features

- **Address**
  - Locate a given word:
    - By position in stack
    - By position in buffer
    - By attachment to a word in buffer

- **Attributes**
  - Identity of word
  - Lemma for word
  - POS tag of word
  - Dependency label for word ← *conditioned on previous decisions!"
## Example:

<table>
<thead>
<tr>
<th>Action</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ROOT]</td>
<td>[They told him a story]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, They]</td>
<td>[told him a story]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, They, told]</td>
<td>[him a story]</td>
</tr>
<tr>
<td>Left-Arc (subj)</td>
<td>[ROOT, told]</td>
<td>[him a story]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, told, him]</td>
<td>[a story]</td>
</tr>
<tr>
<td>Right-Arc (iobj)</td>
<td>[ROOT, told]</td>
<td>[a story]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, told, a]</td>
<td>[story]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, told, a, story]</td>
<td>[]</td>
</tr>
<tr>
<td>Left-Arc (Det)</td>
<td>[ROOT, told, story]</td>
<td>[]</td>
</tr>
<tr>
<td>Right-Arc (dobj)</td>
<td>[ROOT, told]</td>
<td>[]</td>
</tr>
<tr>
<td>Right-Arc (root)</td>
<td>[ROOT]</td>
<td>[]</td>
</tr>
</tbody>
</table>

The diagram illustrates the subject (`subj`), indirect object (`iobj`), direct object (`dobj`), and determiner (`det`) relationships in the sentence "They told him a story."
Transition-Based Parsing

Summary

● **Shift-Reduce** \([\text{reduce} = \text{pop}]\) paradigm, bottom-up approach

● **Pros:**
  ● Single pass, \(O(n)\) complexity
  ● Reduce parsing to classification problem; easy to introduce new features

● **Cons:**
  ● Only makes local decisions, may not find global optimum
  ● Does not handle non-projective trees without hacks
    ● e.g. transforming nonprojective trees to projective in training data; reconverting after
Other Notes

• …is this a parser?
  • No, not really!
  • Transforms problem into sequence labeling task, of a sort.
    • e.g. (SH, LA, SH, RA, SH, SH, LA, RA)
    • Sequence score is sum of transition scores
Other Notes

- Classifier: Any
- Originally, SVMs
- Currently: NNs (LSTMs, pre-trained Transformer-based)
- State-of-the-art: UAS: 97.2%; LAS: 95.7%
Announcing SyntaxNet: The World’s Most Accurate Parser Goes Open Source
Thursday, May 12, 2016

At Google, we spend a lot of time thinking about how computer systems can read and understand human language in order to process it in intelligent ways. Today, we are excited to share the fruits of our research with the broader community by releasing SyntaxNet, an open-source neural network framework implemented in TensorFlow that provides a foundation for Natural Language Understanding (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as Parsey McParseface, an English parser that we have trained for you and that you can use to analyze English text.

Parsey McParseface is built on powerful machine learning algorithms that learn to analyze the linguistic structure of language, and that can explain the functional role of each word in a given sentence. Because Parsey McParseface is the most accurate such model in the world, we hope that it will be useful to developers and researchers interested in automatic extraction of information, translation, and other core applications of NLU.

https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html
Google just open sourced something called ‘Parsey McParseface,’ and it could change AI forever

by Nate Swanner — May 12, 2016 in Design & Dev
Great paper

Many methodological lessons on how to improve transition-based dependency parsing

BUT: don’t believe (or at least beware) the hype!
Dependency Parsing: Summary

● Dependency Grammars:
  ● Compactly represent pred–arg structure
  ● Lexicalized, localized
  ● Natural handling of flexible word order

● Dependency parsing:
  ● Conversion to phrase structure trees
  ● Graph-based parsing (MST), efficient non-proj $O(n^2)$
  ● Transition-based parser
    ● MALTparser: very efficient $O(n)$
    ● Optimizes local decisions based on many rich features
Roadmap

- Dependency Parsing
  - Transition-based Parsing
- Feature-based Parsing
  - Motivation
  - Features
  - Unification
Feature-Based Parsing
Constraints & Compactness

● S → NP VP
  ● They run.
  ● He runs.

● But…
  ● *They runs
  ● *He run
  ● *He disappeared the flight

● Violate agreement (number/person), subcategorization -> over-generation
Enforcing Constraints with CFG Rules

- Agreement
  - $S \rightarrow NP_{sg+3p} \ VP_{sg+3p}$
  - $S \rightarrow NP_{pl+3p} \ VP_{pl+3p}$

- Subcategorization:
  - $VP \rightarrow V_{transitive} \ NP$
  - $VP \rightarrow V_{intransitive}$
  - $VP \rightarrow V_{ditransitive} \ NP \ NP$

- Explosive, and loses key generalizations
Feature Grammars

- Need compact, general constraint

- $S \rightarrow NP \ VP$ [iff NP and VP agree]

- How can we describe agreement & subcategory?
  - Decompose into elementary features that must be consistent
    - e.g. Agreement on number, person, gender, etc

- Augment CF rules with feature constraints
  - Develop mechanism to enforce consistency
  - Elegant, compact, rich representation
Feature Representations

- Fundamentally **Attribute-Value pairs**
  - Values may be symbols or feature structures
  - Feature path: list of features in structure to value
  - “Reentrant feature structure” — sharing a structure

- Represented as
  - Attribute-Value Matrix (AVM)
  - Directed Acyclic Graph (DAG)
Attribute-Value Matrices (AVMs)

\[
\begin{bmatrix}
\text{ATTRIBUTE}_1 & \text{value}_1 \\
\text{ATTRIBUTE}_2 & \text{value}_2 \\
\vdots \\
\text{ATTRIBUTE}_n & \text{value}_n
\end{bmatrix}
\]
AVM Examples

(A) [NUMBER PL PERSON 3]

(B) [CAT NP NUMBER PL PERSON 3]

(C) [CAT AGREEMENT NP NUMBER PL PERSON 3]

(D) [CAT S AGREEMENT 1 NUMBER PL PERSON 3 AGREEMENT 1]

[SUBJECT]
AVM vs. DAG

CAT AGREEMENT

NP

NUMBER PL PERSON 3

CAT

NP

NUMBER

AGREEMENT

PERSON

3rd
Using Feature Structures

- Feature Structures provide formalism to specify constraints
- …but how to apply the constraints?
- Unification
Unification:

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if:
  - Feature structures *match where both have values*
  - Feature structures *differ only where one value is missing or underspecified*
    - Missing or underspecified values are filled with constraints of other

- Result of unification incorporates constraints of both
Subsumption

- Less specific feature structure *subsumes* more specific feature structure
- FS \( F \) subsumes FS \( G \) iff:
  - For every feature \( x \) in \( F \), \( F(x) \) subsumes \( G(x) \)
  - for all paths \( p \) and \( q \) in \( F \) s.t. \( F(p) = F(q) \), \( G(p) = G(q) \)
- Examples:
  - \( A = \begin{bmatrix} \text{NUMBER SG} \\ \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix} \)
  - \( B = \begin{bmatrix} \text{PERSON 3} \end{bmatrix} \)
  - \( C = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix} \)
  - \( A \) subsumes \( C \)
  - \( B \) subsumes \( C \)
  - \( B \) & \( A \) *don’t subsume*
Unification Examples

- **Identical**
  \[ \text{NUMBER SG} \cup \text{NUMBER SG} = \text{NUMBER SG} \]

- **Underspecified**
  \[ \text{NUMBER SG} \cup \_ = \text{NUMBER SG} \]

- **Different Specs**
  \[ \text{NUMBER SG} \cup \text{PERSON 3} = \text{NUMBER SG\ PERSON 3} \]

- **Conflicting Specs**
  \[ \text{NUMBER SG} \cup \text{NUMBER PL} = \emptyset \]
Larger Unification Example

\[
\begin{array}{c}
\text{AGREEMENT } \begin{bmatrix} 1 \\ \text{SUBJECT} \end{bmatrix}
\cup
\begin{bmatrix} 1 \\ \text{AGREEMENT} \end{bmatrix}
\begin{bmatrix} 1 \\ \text{SUBJECT} \end{bmatrix}
\begin{bmatrix} 1 \\ \text{AGREEMENT} \end{bmatrix}
\begin{bmatrix} \text{PERSON } 3 \\ \text{NUMBER } \text{SG} \end{bmatrix}
= \\
\end{array}
\]
One More Unification Example
Unification
Rule Representation

- \( \beta \rightarrow \beta_1 \ldots \beta_n \) (set of constraints) \( \langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle \)
- \( PRON \rightarrow \text{‘he’} \)

\( \langle PRON \ \text{AGREEMENT \ PERSON} \rangle = 3rd \)

Pron \[\text{AGREEMENT} \]

\[\text{PERSON} \rightarrow 3rd\]
Rule Representation

- $\beta \rightarrow \beta_1 \ldots \beta_n$
  \{set of constraints\} \hspace{1cm} $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$

- $NP \rightarrow PRON$

$\langle NP \text{ AGREEMENT PERSON} \rangle = \langle PRON \text{ AGREEMENT PERSON} \rangle$

Diagram:
- NP<br>ACTION<br>PERSON
- PRON<br>ACTION<br>PERSON

“unifiable”
Agreement with Heads and Features

- $\beta \rightarrow \beta_1 \ldots \beta_n$
  
  \{set of constraints\} \hspace{1cm} $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$

$S \rightarrow NP \ VP$

$\langle NP \ \text{AGREEMENT} \rangle = \langle VP \ \text{AGREEMENT} \rangle$

$S \rightarrow Aux \ NP \ VP$

$\langle Aux \ \text{AGREEMENT} \rangle = \langle NP \ \text{AGREEMENT} \rangle$

$NP \rightarrow Det \ Nominal$

$\langle Det \ \text{AGREEMENT} \rangle = \langle Nominal \ \text{AGREEMENT} \rangle$

$\langle NP \ \text{AGREEMENT} \rangle = \langle Nominal \ \text{AGREEMENT} \rangle$

$Aux \rightarrow does$

$\langle Aux \ \text{AGREEMENT \ Number} \rangle = sg$

$\langle Aux \ \text{AGREEMENT \ Person} \rangle = 3rd$

$Det \rightarrow this$

$\langle Det \ \text{AGREEMENT \ Number} \rangle = sg$

$Det \rightarrow these$

$\langle Det \ \text{AGREEMENT \ Number} \rangle = pl$

$Verb \rightarrow serve$

$\langle Verb \ \text{AGREEMENT \ Number} \rangle = pl$

$Noun \rightarrow flight$

$\langle Noun \ \text{AGREEMENT \ Number} \rangle = sg$
Simple Feature Grammars in NLTK

- $S \rightarrow NP\ VP$
Simple Feature Grammars

- $S \rightarrow NP[NUM=?n] \ VP[NUM=?n]$
- $NP[NUM=?n] \rightarrow N[NUM=?n]$
- $NP[NUM=?n] \rightarrow \text{PropN[NUM=?n]}$
- $NP[NUM=?n] \rightarrow \text{Det[NUM=?n]} \ N[NUM=?n]$
- $\text{Det[NUM=sg]} \rightarrow 'this' | 'every'$
- $\text{Det[NUM=pl]} \rightarrow 'these' | 'all'$
- $N[NUM=sg] \rightarrow 'dog' | 'girl' | 'car' | 'child'$
- $N[NUM=pl] \rightarrow 'dogs' | 'girls' | 'cars' | 'children'$
Parsing with Features

```python
>>> cp = load_parser('grammars/book_grammars/feat0.fcfg')
>>> for tree in cp.parse(tokens):
...     print(tree)

(S[] (NP[NUM='sg']
    (PropN[NUM='sg'] Kim))
    (VP[NUM='sg', TENSE='pres']
        (TV[NUM='sg', TENSE='pres'] likes)
        (NP[NUM='pl'] (N[NUM='pl'] children))))
```
Feature Applications

● Subcategorization
  ● Verb-Argument constraints
    ● Number, type, characteristics of args
    ● e.g. is the subject *animate*?
    ● Also adjectives, nouns

● Long-distance dependencies
  ● e.g. filler–gap relations in wh-questions
  ● “Which flight do you want me to have the travel agent book?”
Morphosyntactic Features

● Grammatical feature that influences morphological or syntactic behavior

● English:
  ● Number:
    ● Dog, dogs
  ● Person:
    ● am; are; is
  ● Case:
    ● I / me; he / him; etc.
Semantic Features

● Grammatical features that influence semantic (meaning) behavior of associated units

● E.g.:
  ● ?The rocks slept.

● Many proposed:
  ● Animacy: +/-
  ● Gender: masculine, feminine, neuter
  ● Human: +/-
  ● Adult: +/-
  ● Liquid: +/-
Aspect (J&M 17.4.2)

- The climber [hiked] [for six hours].
- The climber [hiked] [on Saturday].
- The climber [reached the summit] [on Saturday].
- *The climber [reached the summit] [for six hours].

Contrast:
- Achievement (in an instant) vs activity (for a time)