Alternative Syntactic Theories

L614

Spring 2015
Syntactic analysis

- **Generative grammar**: collection of words and rules with which we generate strings of those words, i.e., sentences.

- **Syntax** attempts to capture the nature of those rules.
  1. Colorless green ideas sleep furiously.
  2. *Furiously sleep ideas green colorless.

- What generalizations are needed to capture the difference between grammatical sentences and ungrammatical sentences?

Using a particular formalism, a theory encapsulates these generalizations:

- i.e., a theory is a grammar.
Formalism vs. Theory

Will we actually look at theories? ... Sort of.

- A **theory** describes a set of data and makes predictions for new data
  - In this class, we will emphasize theories which are *testable*, i.e., can be verified or falsified

- A **formalism** provides a way of defining a theory with mathematical rigor
  - It is essentially a set of beliefs and conditions that frame how generalizations can be made.

The course name (*Alternative Syntactic Theories*) is a bit of a misnomer: we will actually be focusing on *formalisms*, and we will use theories to exemplify them.
Roughly speaking, **transformational syntax** (GB, P&P, ...) has focused on the following:

- **Explanatory adequacy**: does the theory fit with a deeper model (e.g., universal grammar)?
- **Psychological modeling**: does the grammar make sense in light of what we know of how the mind works?
- **Universality**: are the generalizations applicable to all languages?
- **Transformations/Movement**: are (surface) sentences derived from underlying sentences? (e.g. passives from actives)

These kinds of theories have not generally been integrated with computational applications.
Making it computational

How can grammatical theories be useful for computational linguistics?

- Parsing: take an input sentence and return the syntactic analysis and/or state whether it is a valid sentence
- Generation: take a meaning representation and generate a valid sentence

⇒ Both tasks are often subparts of practical applications (e.g., dialogue systems)
  - Both can also provide feedback to the grammar writer
To use a grammar for parsing or generation, we need to have a grammar that meets several criteria:

- Accurate: gives a correct analysis
- Precise: tells a computer exactly what to do
- Efficient: able to parse a sentence and return one or only a small number of parses
- Useful: is relatively easy to map a syntactic structure of a sentence to its meaning

⇒ Not necessarily why computational formalisms were developed, but the formalisms enable such uses
The formalisms we will look generally share several properties:

- Descriptively adequate
- Precisely encoded (implementable)
- Constrained in the mathematical formalism
- Monostratal
- (Usually) highly lexical
Descriptively adequate

One could explain the underlying mechanisms, but we are mostly concerned with being able to *describe* linguistic phenomena

- Provide a structural description for every well-formed sentence
  - Define which sentences are well-formed in a language & which not
- Give an accurate encoding of a language
- Broad-coverage: describe all of a language
  - Less of a distinction between core & periphery phenomena
Precisely encoded

Mathematical formalism: formal way to generate sets of strings

Thus, we need to precisely define:

- elementary structures
- ways of combining those structures

Such an emphasis on mathematical precision makes these grammar formalisms more easily implementable

- e.g., can answer the question of whether different parts of a grammar will conflict
Constrained in the mathematical formalism

Formalism should (arguably) be **constrained**, i.e., cannot be allowed to specify all strings

- **Linguistic motivation**: Limit the scope of the theory of grammar
- **Computational motivation**: Allow one to define efficient processing models

This is different than constraining a theory

- **What is the minimum amount of mathematical overhead that we need to describe language?**
Only have one (surface) syntactic level

- Make no recourse to movement or transformations
- Augment your basic (phrase structure) tree with information that can describe “movement” phenomena
  - Need some way to relate different structures (e.g., active and passive) without invoking, e.g., traces

⇒ Without having to refer to movement, easier to process sentences computationally
Lexical

Some approaches: rules apply to broad classes & only some information in the lexicon (e.g., subcategorization)

But more and more theories emphasize the role of individual lexical items in grammatical constructions

- Linguistic motivation: lexicon best way to specify some generalizations: *He told/me the truth*
- Computational motivation: lexical information can be derived from corpora

⇒ Shift more of the information to the lexicon; each lexical item is thus a complex object
Brief mention of complexity

We have touched on the complexity of different formalisms

<table>
<thead>
<tr>
<th>Type</th>
<th>Automaton</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Memory</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Unbounded</td>
<td>TM</td>
</tr>
<tr>
<td>1</td>
<td>Bounded</td>
<td>LBA</td>
</tr>
<tr>
<td>2</td>
<td>Stack</td>
<td>PDA</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
<td>FSA</td>
</tr>
</tbody>
</table>

- TM: Turing Machine
- LBA: Linear-Bounded Automaton
- PDA: Push-Down Automaton
- FSA: Finite-State Automaton
Criteria under which to evaluate grammar formalisms

Three kinds of criteria:
- linguistic naturalness
- mathematical power
- computational effectiveness and efficiency

The weaker the type of grammar:
- the stronger the claim made about possible languages
- the greater the potential efficiency of the parsing procedure

Reasons for choosing a stronger grammar class:
- to capture the empirical reality of actual languages
- to provide for elegant analyses capturing more generalizations (e.g., more “compact” grammars)
Context-Free Grammars (CFGs): probably the most popular formalism for writing English grammars

- elementary structures: rules composed of nonterminal and terminal elements
- combine rules by rewriting them

Example of a set of rules:

- S → NP VP
- NP → Det N
- VP → V NP
- ...

Empirical downside: the rules are rather impoverished ...
Are CFGs good enough?

- Data from Swiss German & other languages show that CFGs are not powerful enough to handle all natural language constructions.
- CFGs are not easily lexicalized.
- CFGs become complicated once we start taking into account agreement features, verb subcategorizations, unbounded dependency constructions, raising constructions, etc.

We need more refined formalisms ...
We want to move beyond CFGs to better capture language, but maintain that level of precision.

One can view this in different ways:

- Extend the basic model of CFGS with, e.g., complex categories, functional structure, feature structures, ...
- Eliminate CFG model or derive it some other way

The frameworks we will investigate explore different ways of looking at syntax ...
Computational Grammar Frameworks

What we will look at the rest of the semester:

- Dependency Grammar (DG)
- Tree-Adjoining Grammar (TAG)
- Lexical-Functional Grammar (LFG)
- Head-driven Phrase Structure Grammar (HPSG)
- Combinatory Categorial Grammar (CCG)
Dependency Grammar (DG)

- The way to analyze a sentence is by looking at the relations between words
- Generally speaking, no grouping (constituency) is used
  - DG is not a unified framework; there are a host of different frameworks within this tradition
  - DG bears similarity to functional structure, but have often been derived independent of CFG traditions
- Analyses tend to be closely related to the semantics of a sentence

Some frameworks we’ll investigate utilize insights from DG
Tree-Adjoining Grammar (TAG)

Roughly: analysis looks like a CFG tree, but the way to obtain it is different

- Elementary structures are trees of arbitrary height
- Trees are rooted in lexical items, i.e. lexicalized
  - In other words, the lexicon contains tree fragments as parts of lexical entries
- Put trees together by substituting and adjoining them, resulting in a final tree which looks like a CFG-derived tree
Lexical-Functional Grammar (LFG)

- Functional structure (subject, object, etc.) divided from constituent structure (tree structure)
  - Akin to dependency structure + phrase structure
  - The f-structures are potentially very complex
- Can express some generalizations in f-structure; some in c-structure;
  - i.e., not restricted to saying everything in terms of trees
Head-driven Phrase Structure Grammar (HPSG)

- Sentences, phrases, & words all uniformly treated as linguistic signs, i.e., complex objects of features
  - Analyses can rely on CFG backbone, but need not
- Similar to LFG in its use of a feature architecture
- Uses inheritance hierarchy to relate different objects
  - e.g., nouns and determiners are both types of nominals
Combinatory Categorial Grammar (CCG)

- Categorial Grammar derives sentences in a proof-solving manner
  - Maintains close link with semantic representation
- Lexical categories specify how to combine words into sentences
  - Again, lexical entries contain tree-like information
- CCG has sophisticated mechanisms to deal with coordination, extraction, & other constructions