Alternative Syntactic Theories

L614
Spring 2010

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?

Syntax: What does it mean?

Can view a syntactic theory in a number of ways, two of which are the following:

- **Psychological model**: syntactic structures correspond to what is in the heads of speakers
- **Computational model**: syntactic structures are formal objects which can be mathematically manipulated.

⇒ We will focus on the computational way of viewing grammar for this class

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.

The Transformational tradition

Roughly speaking, **transformational syntax** (GB, P&P, ...) has focused on the following:

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

But this kind of theory often doesn’t lend itself well to computational applications

Alternative assumptions

- Prioritize descriptive adequacy over explanatory adequacy
- Prioritize computational effectiveness over psychological reality
  - e.g., movement is disfavored
- Prioritize description in one language before dealing with all languages

The data will always be the same, but how you handle it, as we’ll see, depends largely upon your assumptions
Making it computational

How is a grammatical theory 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)

Computational needs

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 it is that you want it 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

⇒ These needs are not necessarily why the computational formalisms were developed, but they are some of the reasons why people use them.

Computational Grammar Formalisms

The formalisms we will look at this quarter generally share several properties:

- Descriptive adequacy
- Precise encodings (implementable)
- Constrained mathematical formalisms
- Monostratal frameworks
- (Usually) highly lexical

Descriptive adequacy

Some researchers try to explain the underlying mechanisms, but we are most concerned with being able to describe linguistic phenomena

- Provide a structural description for every well-formed sentence
  - Define which sentences are well-formed and which are not in a language
- Give us an accurate encoding of a language
- Interested in broad-coverage, i.e., can (try to) describe all of a language
  → less of a distinction between core and periphery phenomena

Precise encodings

Mathematical formalism: formal way to generate sets of strings
Precisely define:
- elementary structures
- ways of combining those structures

⇒ Such an emphasis on mathematical precision makes these grammar formalisms more easily implementable
- Will 2 parts of your grammar conflict?
- If we have precisely encoded the grammar, we can answer this question with certainty.

Constrained mathematical formalisms

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

- Linguistic motivation: Limits the scope of the theory of grammar
- Computational motivation: Allows us 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?
Monostratal frameworks

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 on a computer.

Lexical

In the past, rules applied to broad classes and only some information was put in the lexicon, e.g., subcategorization information.

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/*divulged me the truth
- Computational motivation: can derive lexical information from corpora

⇒ Shift more of the information to the lexicon; each lexical item may be a complex object.

Brief mention of complexity

We have touched on the complexity of different formalisms

<table>
<thead>
<tr>
<th>Type</th>
<th>Automation</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unbounded</td>
<td>Memory, TM, α → β General rewrite</td>
</tr>
<tr>
<td>1</td>
<td>Bounded</td>
<td>LBA, β A γ → β δ γ Context-sensitive</td>
</tr>
<tr>
<td>2</td>
<td>Stack</td>
<td>PDA, A → δ Context-free</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
<td>FSA, A → x Right linear</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

There are 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 (→ more “compact” grammars)

Context-Free Grammars (CFGs)

Context-Free Grammars (CFGs) are one kind of constrained mathematical formalism and probably the most popular 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
- ...

But these rules are rather impoverished ...

Are CFGs good enough?

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

We need more refined formalisms ...
Beyond CFGs

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

We can look at it a couple of 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 take one of these approaches.

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
- No grouping, or constituency, is used
  - DG traditions are often completely independent of constituency-based traditions (e.g., CFGs)
  - DG is not a unified framework; there are a host of different frameworks within this tradition
- A verb and its arguments drive an analysis, which is closely related to the semantics of a sentence

Some of the other frameworks we'll investigate utilize insights from DG

Tree-Adjoining Grammar (TAG)

The analysis looks like a CFG tree, but the way to get it is completely 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)
  - Kind of like combining dependency structure with phrase structure
  - The f-structures are potentially very complex, however.
- 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, and words all uniformly treated as linguistic signs, i.e., complex objects of features
  - Many analyses rely on a CFG backbone, but this need not be so.
- Similar to LFG in its use of a feature architecture
- Uses an inheritance hierarchy to relate different types of objects (e.g., nouns and determiners are both types of nominals)
Combinatory Categorial Grammar (CCG)

- Categorial Grammar derives sentences in a proof-solving manner, maintaining a close link with a semantic representation

- Lexical categories specify how to combine words into sentences
  - The idea of selection is crucial, e.g., a verb will select for the number and type of arguments
  - Again, lexical entries contain tree-like information

- CCG has sophisticated mechanisms that deal nicely with coordination, extraction, and other constructions