Lecture 20: Expressive grammars

Projects and Literature Reviews

First report due Nov 15
(PDF written in LaTeX; no length restrictions; submission through Compass)

Purpose of this first report:
Check-in to make sure that you’re on track
(or, if not, that we can spot problems)
Get feedback from your peers

Rubrics for the final reports (due on Reading Day):
https://courses.engr.illinois.edu/CS447/LiteratureReviewRubric.pdf
https://courses.engr.illinois.edu/CS447/FinalProjectRubric.pdf

Projects and Literature Reviews

Guidelines for first Project Report:
What is your project about?
What are the relevant papers you are building on?
What data are you using?
What evaluation metric will you be using?
What models will you implement/evaluate?
What is your to-do list?

Guidelines for first Literature Review Report:
What is your literature review about?
(What task or what kind of models?)
What are the papers you will review?
(If you already have it, give a brief summary of each of them)
What’s your to-do list?

Why grammar?

Surface string
Mary saw John

Grammar

Parsing

Logical form:
saw(Mary,John)

Pred-arg structure:
PRED saw
AGENT Mary
PATIENT John

Dependency graph:
saw
Mary
John
Grammar formalisms

Formalisms provide a language in which linguistic theories can be expressed and implemented.

Formalisms define elementary objects (trees, strings, feature structures) and recursive operations which generate complex objects from simple objects.

Formalisms may impose constraints (e.g. on the kinds of dependencies they can capture).

How do grammar formalisms differ?

Formalisms define different representations:
- Tree-joining Grammar (TAG): Fragments of phrase-structure trees
- Lexical-functional Grammar (LFG): Annotated phrase-structure trees (c-structure) linked to feature structures (f-structure)
- Combinatory Categorial Grammar (CCG): Syntactic categories paired with meaning representations
- Head-Driven Phrase Structure Grammar (HPSG): Complex feature structures (Attribute-value matrices)

The dependencies so far:

Arguments:
Verbs take arguments: subject, object, complements, ...
Heads subcategorize for their arguments

Adjuncts/Modifiers:
Adjectives modify nouns, adverbs modify VPs or adjectives, PPs modify NPs or VPs
Modifiers subcategorize for the head

Typically, these are local dependencies: they can be expressed within individual CFG rules

Context-free grammars

CFGs capture only nested dependencies
The dependency graph is a tree
The dependencies do not cross

VP \rightarrow \text{Adv Verb NP}
Beyond CFGs: Nonprojective dependencies

Dependencies form a tree with crossing branches

(Non-local) scrambling: In a sentence with multiple verbs, the argument of a verb appears in a different clause from that which contains the verb (arises in languages with freer word order than English)

Die Pizza hat Klaus versprochen zu bringen
The pizza has Klaus promised to bring
Klaus has promised to bring the pizza

Extraposition: Here, a modifier of the subject NP is moved to the end of the sentence

The guy is coming who is wearing a hat
Compare with the non-extraposed variant
The [guy [who is wearing a hat]] is coming

Topicalization: Here, the argument of the embedded verb is moved to the front of the sentence.

Cheeseburgers, I [thought [he likes]]

Beyond CFGs: Nonlocal dependencies

Dependencies form a DAG (a node may have multiple incoming edges)

Arise in the following constructions:
- Control (He has promised me to go), raising (He seems to go)
- Wh-movement (the man who you saw yesterday is here again),
- Non-constituent coordination (right-node raising, gapping, argument-cluster coordination)

Dependency structures

Nested (projective) dependency trees (CFGs)

Non-projective dependency trees

Non-local dependency graphs
Non-local dependencies

Unbounded nonlocal dependencies

Wh-questions and relative clauses contain unbounded nonlocal dependencies, where the missing NP may be arbitrarily deeply embedded:

‘the sushi that [you told me [John saw [Mary eat]]]’

‘what [did you tell me [John saw [Mary eat]]]?’

Linguists call this phenomenon wh-extraction (wh-movement).

Long-range dependencies

Bounded long-range dependencies:
Limited distance between the head and argument

Unbounded long-range dependencies:
Arbitrary distance (within the same sentence) between the head and argument

Unbounded long-range dependencies cannot (in general) be represented with CFGs.

Chomsky’s solution:
Add null elements (and co-indexation)
The trace analysis of *wh*-extraction

Because only one element can be extracted, we can use **slash categories**. This is still a CFG: the set of nonterminals is finite.

---

**German: center embedding**

...daß ich [Hans Schwimmen] sah
...that I Hans saw
...that I saw [Hans swim]

...daß ich [Maria [Hans schwimmen] helfen] sah
...that I Maria Hans swim help saw
...that I saw [Mary help [Hans swim]]

---

**Dutch: cross-serial dependencies**

...dat ik Hans zag zwemmen
...that I Hans saw swim
...that I saw [Hans swim]

...dat ik Maria Hans zag helpen zwemmen
...that I Maria Hans saw help swim
...that I saw [Mary help [Hans swim]]

...dat ik Anna Maria Hans zag laten helpen zwemmen
...that I Anna Maria Hans saw let help swim
...that I saw [Anna let [Mary help [Hans swim]]]

Such **cross-serial dependencies require mildly context-sensitive grammars**
Two mildly context-sensitive formalisms: TAG and CCG

Mildly context-sensitive grammars

Contain all context-free grammars/languages

Can be parsed in polynomial time (TAG/CCG: O(n^6))

(Strong generative capacity) capture certain kinds of dependencies: nested (like CFGs) and cross-serial (like the Dutch example), but not the MIX language:

MIX: the set of strings \( w \in \{a, b, c\}^* \) that contain equal numbers of \( a \)s, \( b \)s and \( c \)s

Have the constant growth property:
the length of strings grows in a linear way
The power-of-2 language \( \{a^{2^n}\} \) does not have the constant growth property.

The Chomsky Hierarchy

Recursively enumerable
Context-sensitive
Mildly context-sensitive
Context-free
Regular

TAG and CCG are lexicalized formalisms

The lexicon:
- pairs words with elementary objects
- specifies all language-specific information
  (e.g. subcategorization information)

The grammatical operations:
- are universal
- define (and impose constraints on) recursion.
Tree-Adjoining Grammar

(Lexicalized) Tree-Adjoining Grammar

TAG is a tree-rewriting formalism:
TAG defines operations (substitution, adjunction) on trees. The elementary objects in TAG are trees (not strings)

TAG is lexicalized:
Each elementary tree is anchored to a lexical item (word) “Extended domain of locality”:
The elementary tree contains all arguments of the anchor.
TAG requires a linguistic theory which specifies the shape of these elementary trees.

TAG is mildly context-sensitive:
can capture Dutch cross-serial dependencies but is still efficiently parseable

Extended domain of locality

We want to capture all arguments of a word in a single elementary object.

We also want to retain certain syntactic structures (e.g. VPs).

Our elementary objects are tree fragments:

\[
\text{S} \rightarrow \text{NP} \quad \text{VP} \\
\text{NP} \rightarrow \text{VBZ} \quad \text{NP} \\
\text{eats}
\]

TAG substitution (arguments)

Substitute

Derived tree:

\[
\alpha_1: \\
\alpha_2: \quad \alpha_3: \\
\text{X}_{\alpha_1} \quad \text{Y}_{\alpha_1}
\]

Derivation tree:

\[
\alpha_2 \quad \alpha_3
\]
The effect of adjunction

- **No adjunction:** TSG (Tree substitution grammar)
  - TSG is context-free

- **Sister adjunction:** TIG (Tree insertion grammar)
  - TIG is also context-free, but has a linguistically more adequate treatment of modifiers

- **Wrapping adjunction:** TAG (Tree-adjoining grammar)
  - TAG is mildly context-sensitive

A small TAG lexicon

- **α2:**
  - NP
  - John

- **α3:**
  - NP
  - tapas

A TAG derivation
A TAG derivation

\[ S \rightarrow NP \quad VP \]

\[ \alpha_1 \quad \alpha_2 \quad \beta_1 \quad \alpha_3 \]

\[ NP \rightarrow John \quad VBZ \quad NP \quad eats \quad tapas \]

\[ \beta_1 \quad VP \quad VP^* \]

\[ RB \quad always \]

\[ S \rightarrow a \quad b \]

\[ a^n b^n: \text{Cross-serial dependencies} \]

Elementary trees:

\[ S \]

Deriving \( aabb \)

Combinatory Categorial Grammar
CCG: the machinery

Categories:
specify subcat lists of words/constituents.

Combinatory rules:
specify how constituents can combine.

The lexicon:
specifies which categories a word can have.

Derivations:
spell out process of combining constituents.

CCG categories

Simple (atomic) categories: NP, S, PP

Complex categories (functions):
Return a result when combined with an argument

<table>
<thead>
<tr>
<th>Function</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP, intransitive verb</td>
<td>S\NP</td>
</tr>
<tr>
<td>Transitive verb</td>
<td>(S\NP)/NP</td>
</tr>
<tr>
<td>Adverb</td>
<td>(S\NP)(S\NP)</td>
</tr>
<tr>
<td>Prepositions</td>
<td>((S\NP)(S\NP))/NP</td>
</tr>
<tr>
<td></td>
<td>(NP\NP)/NP</td>
</tr>
<tr>
<td></td>
<td>PP/NP</td>
</tr>
</tbody>
</table>

Function application

Forward application (>):

(\S\NP)/NP NP => S\NP
        eats       tapas       eats tapas

Backward application (<):

NP S\NP =>< S
        John eats tapas       John eats tapas

Used in all variants of categorial grammar

A (C)CG derivation
Function composition

**Harmonic forward composition (>B):**
\[ \frac{X}{Y} \quad \frac{Y}{Z} \Rightarrow_{>B} \frac{X}{Z} \]

**Harmonic backward composition (<B):**
\[ \frac{Y}{Z} \quad \frac{X}{Y} \Rightarrow_{<B} \frac{X}{Z} \]

**Forward crossing composition (>B_x):**
\[ \frac{X}{Y} \quad \frac{Y}{Z} \Rightarrow_{>B_x} \frac{X}{Z} \]

**Backward crossing composition (<B_x):**
\[ \frac{Y}{Z} \quad \frac{X}{Y} \Rightarrow_{<B_x} \frac{X}{Z} \]

Type-raising

**Forward type-raising (>T):**
\[ X \Rightarrow_{>T} \frac{T}{T\setminus X} \]

**Backward type-raising (<T):**
\[ X \Rightarrow_{<T} \frac{T}{T\setminus X} \]

Type-raising and composition

**Type-raising:** \( X \rightarrow \frac{T}{T\setminus X} \)
- Turns an argument into a function.
- NP \rightarrow \frac{S}{S\setminus NP} \quad \text{(subject)}
- NP \rightarrow \frac{S\setminus NP}{(S\setminus NP)/NP} \quad \text{(object)}

**Harmonic composition:** \( \frac{X}{Y} \quad \frac{Y}{Z} \rightarrow \frac{X}{Z} \)
- Composes two functions (complex categories)
  - \( \frac{S\setminus NP}{PP} \rightarrow \frac{PP/NP}{(S\setminus NP)/NP} \)
  - \( \frac{S}{S\setminus NP} \rightarrow \frac{S\setminus NP}{NP} \)

**Crossing function composition:** \( \frac{X}{Y} \quad \frac{Y}{Z} \rightarrow \frac{X}{Z} \)
- Composes two functions (complex categories)
  - \( \frac{S\setminus NP}{S} \rightarrow \frac{SNP}{SNP\setminus NP} \)

Wh-movement (relative clause):

- Right-node raising:

- The tapas ordered Mary

- The tapas John ate and

- The tapas S

- S

NP
Function application (> and <):
\[ \frac{X}{Y} \quad Y \quad \Rightarrow \quad X \]
\[ Y \quad \frac{X}{Y} \quad \Rightarrow \quad X \]

Harmonic composition (>B and <B):
\[ \frac{X}{Y} \quad \frac{Y}{Z} \quad \Rightarrow \quad >B \quad \frac{X}{Z} \]
\[ Y \quad \frac{X}{Y} \quad \Rightarrow \quad <B \quad \frac{X}{Z} \]

Crossing composition (>Bx and <Bx):
\[ \frac{X}{Y} \quad \frac{Y}{Z} \quad \Rightarrow \quad >Bx \quad \frac{X}{Z} \]
\[ Y \quad \frac{X}{Y} \quad \Rightarrow \quad <Bx \quad \frac{X}{Z} \]

Generalized composition (>Bn and <Bn):
\[ \frac{X}{Y} \quad ((\ldots | Z_1 | \ldots)|Z_n) \quad \Rightarrow \quad >Bn \quad ((\ldots | Z_1 | \ldots)|Z_n) \]
\[ \ldots((\ldots | Z_1 | \ldots)|Z_n) \quad \frac{X}{Y} \quad \Rightarrow \quad <Bn \quad ((\ldots | Z_1 | \ldots)|Z_n) \]

Type-raising (>T and <T):
\[ X \quad \Rightarrow \quad >B \quad T / (T \setminus X) \]
\[ X \quad \Rightarrow \quad <B \quad T \setminus (T / X) \]

Dutch cross-serial dependencies

\[
\begin{array}{cccccc}
\text{ik} & \text{Maria} & \text{Hans} & \text{zag} & \text{helpen} & \text{zwimmen} \\
\text{NP} & \text{NP} & \text{NP} & (S \setminus \text{NP}) / S & ((S \setminus \text{NP}) \setminus \text{NP}) / (S \setminus \text{NP}) & S \setminus \text{NP} \\
\end{array}
\]

> Bx

< B

B

Type-raising (>T and <T):
\[ X \quad \Rightarrow \quad >B \quad T / (T \setminus X) \]
\[ X \quad \Rightarrow \quad <B \quad T \setminus (T / X) \]

Combiantory Categorial Grammar

- CCG is **lexicalized**
  (the “rules” of the grammar are completely general, all language-specific information is given in the lexicon)
- CCG is **mildly context-sensitive**
  (can capture Dutch crossing dependencies, but is still efficiently parseable)
- CCG has a **flexible constituent structure**
- CCG has a unified treatment of extraction/coordination
- CCG has a transparent syntax-semantics interface
  (every syntactic category and operation has a semantic counterpart)
- CCG rules are monotonic
  (movement or traces don’t exist)

Today’s key concepts

Phenomena that require extensions of standard context-free grammars:

- non-local dependencies
- cross-serial dependencies

Two lexicalized formalisms:
- Tree-adjoining Grammar
- Combinatory Categorial Grammar