CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

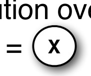
Lecture 8: Formal Grammars of English

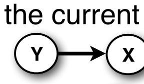
Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

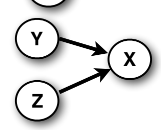
Recap: Wednesday's lecture

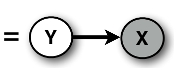
Graphical models for sequence labeling

Directed graphical models

Graphical models are a **notation for probability models**.
In a **directed** graphical model, **each node** represents a distribution over a random variable:
– P(X) = 

Arrows represent dependencies (they define what other random variables the current node is conditioned on)
– P(Y) P(X | Y) = 

– P(Y) P(Z) P(X | Y, Z) = 

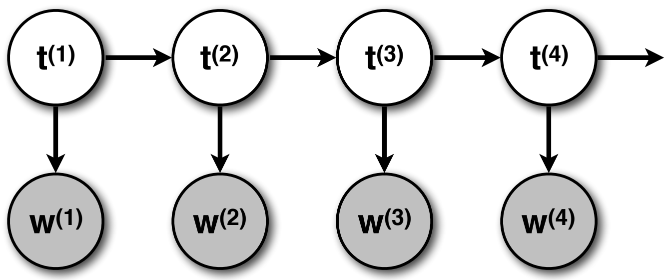
Shaded nodes represent observed variables.
White nodes represent hidden variables
– P(Y) P(X | Y) with Y hidden and X observed = 

HMMs as graphical models

HMMs are **generative** models of the observed input string **w**

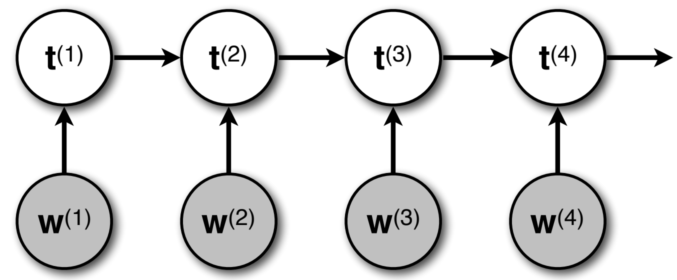
They 'generate' **w** with $P(\mathbf{w},\mathbf{t}) = \prod_i P(t^{(i)} | t^{(i-1)}) P(w^{(i)} | t^{(i)})$

When we use an HMM to tag, we observe **w**, and need to find **t**



Discriminative probability models

A discriminative or **conditional** model of the labels **t** given the observed input string **w** models $P(\mathbf{t} \mid \mathbf{w}) = \prod_i P(t^{(i)} | w^{(i)}, t^{(i-1)})$ directly.



Discriminative models

There are two main types of discriminative probability models:

- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

MEMMs and CRFs:

- are both based on logistic regression
- have the same graphical model
- require the Viterbi algorithm for tagging
- differ in that MEMMs consist of independently learned distributions, while CRFs are trained to maximize the probability of the entire sequence

Probabilistic classification

Classification:

Predict a class (label) c for an input $\mathbf{x}$

There are only a (small) finite number of possible class labels

Probabilistic classification:

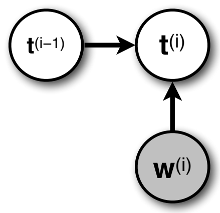
- Model the probability $P(c \mid \mathbf{x})$

  $P(c|\mathbf{x})$ is a probability if $0 \leq P(c_i \mid \mathbf{x}) \leq 1$, and $\sum_i P(c_i \mid \mathbf{x}) = 1$
- Return the class $c^* = \mathrm{argmax}_i \, P(c_i \mid \mathbf{x})$ that has the highest probability

One standard way to model $P(c \mid \mathbf{x})$ is logistic regression (used by MEMMs and CRFs)

Maximum Entropy Markov Models

MEMMs use a MaxEnt classifier for each $P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$:



Since we use w to refer to words, let's use λ_{jk} as the weight for the feature function $f_j(t^{(i-1)}, w^{(i)})$ when predicting tag t_k:

$$P(t^{(i)} = t_k \mid t^{(i-1)}, w^{(i)}) = \frac{\exp(\sum_j \lambda_{jk} f_j(t^{(i-1)}, w^{(i)}))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t^{(i-1)}, w^{(i)}))}$$

Using features

Think of feature functions as useful questions you can ask about the input $\mathbf{x}$:

– **Binary feature functions**:
$f_{\text{first-letter-capitalized}}(\textbf{Urbana}) = 1$
$f_{\text{first-letter-capitalized}}(\textbf{computer}) = 0$

– **Integer (or real-valued) features**:
$f_{\text{number-of-vowels}}(\textbf{Urbana}) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

From features to probabilities

We associate a real-valued weight w_{ic} with each feature function $f_i(\mathbf{x})$ and output class c

Note that the feature function $f_i(\mathbf{x})$ does not have to depend on c as long as the weight does (note the double index w_{ic})

This gives us a real-valued score for predicting class c for input $\mathbf{x}$: $\text{score}(\mathbf{x},c) = \sum_i w_{ic} f_i(\mathbf{x})$

This score could be negative, so we exponentiate it:
$\text{score}(\mathbf{x},c) = \exp(\sum_i w_{ic} f_i(\mathbf{x}))$
To get a probability distribution over all classes c, we renormalize these scores:
$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$
$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$

Learning: finding *w*

Learning = finding weights ***w***
We use conditional maximum likelihood estimation (and standard convex optimization algorithms) to find/learn ***w***
(for more details, attend CS446 and CS546)

The conditional MLE training objective:
Find the **w** that assigns highest probability to all observed outputs c_i given the inputs $\mathbf{x}_i$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_i P(c_i \mid \mathbf{x}_i, \mathbf{w})$$

Terminology

Models that are of the form

$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$

$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$

are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models.

- CS446 and CS546 should give you more details about these.

The normalizing term $\sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$ is also called the partition function and is often abbreviated as Z

Features for Sequence Labeling

What features are useful to model $P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$?

- The identity of the previous label
- Properties of the current word:
  - $w^{(i)}$ starts with/contains a capital letter/number,…
  - $w^{(i)}$ contains the character "A" ("B", …"Z", …1, 2, …0,….)
  - $w^{(i)}$ ends in "ing", "ed", ….
  - …

Feature engineering is essential for any practical

We typically define feature *templates* (e.g. let any of the first, or last, n (=1,2,3,…) characters be used as a separate feature. This results in a very large number of *actual* features (and weights to be learned)

Methods for feature selection become essential

Feature Engineering

Feature engineering (finding useful features) is essential to get good performance out of any classifier

- This requires domain expertise

We typically define feature *templates*

- (e.g. let any of the first, or last, n (=1,2,3,…) characters be used as a separate feature.

This results in a very large number of *actual* features (and weights to be learned)

Methods for feature selection become essential.

On to new material..

Previous key concepts

NLP tasks dealing with words...
- POS-tagging, morphological analysis

… require finite-state representations,
- Finite-State Automata and Finite-State Transducers

… the corresponding probabilistic models,
- Probabilistic FSAs and Hidden Markov Models
- Estimation: relative frequency estimation, EM algorithm

… and appropriate search algorithms
- Dynamic programming: Forward, Viterbi, Forward-Backward

The next key concepts

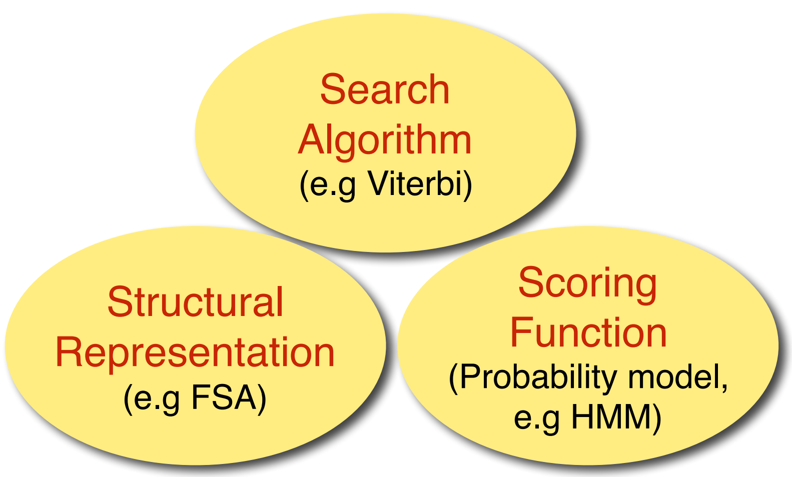
NLP tasks dealing with sentences...
- Syntactic parsing and semantic analysis

… require (at least) context-free representations,
- Context-free grammars, unification grammars

… the corresponding probabilistic models,
- Probabilistic Context-Free Grammars, Loglinear models
- Estimation: Relative Frequency estimation, EM algorithm, etc.

… and appropriate search algorithms
- Dynamic programming: chart parsing, inside-outside algorithm

Dealing with ambiguity

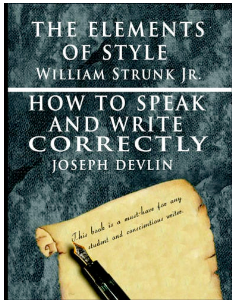


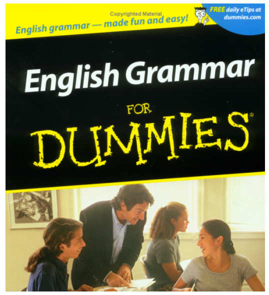
Today's lecture

Introduction to natural language syntax ('grammar'):

Constituency and dependencies
Context-free Grammars
Dependency Grammars
A simple CFG for English

What is grammar?





No, not really, not in this class

What is grammar?

Grammar formalisms
(= linguists' programming languages)

A precise way to define and describe the structure of sentences.

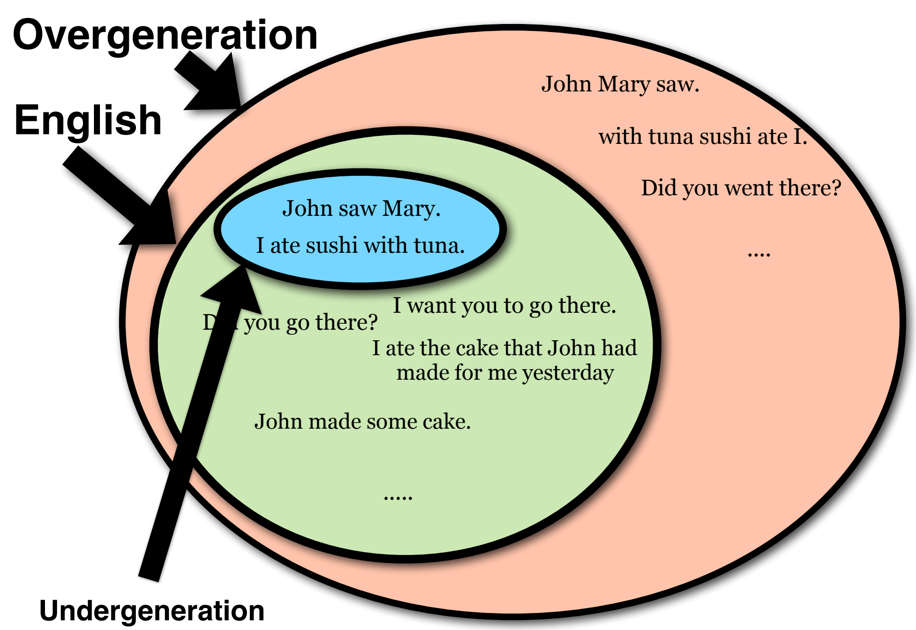
(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

Specific grammars
(= linguists' programs)

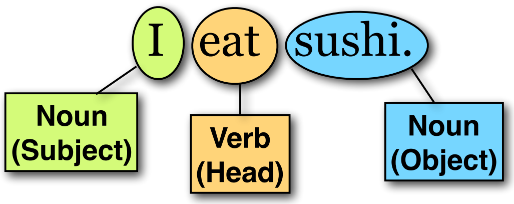
Implementations (in a particular formalism) for a particular language (English, Chinese,....)

Can we define a program that generates all English sentences?

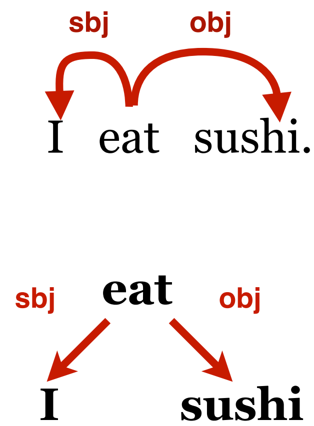
The number of sentences is infinite.
But we need our program to be finite.



Basic sentence structure

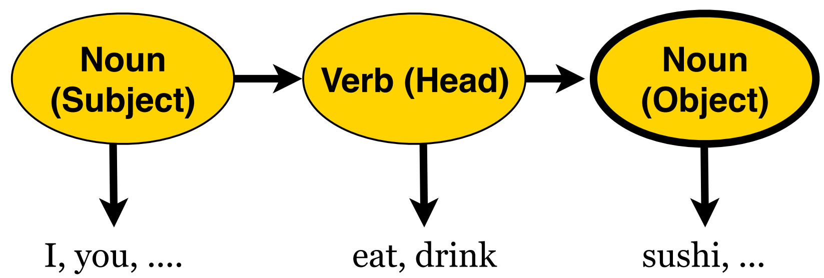


This is a dependency graph:



A finite-state-automaton (FSA)

A Hidden Markov Model (HMM)



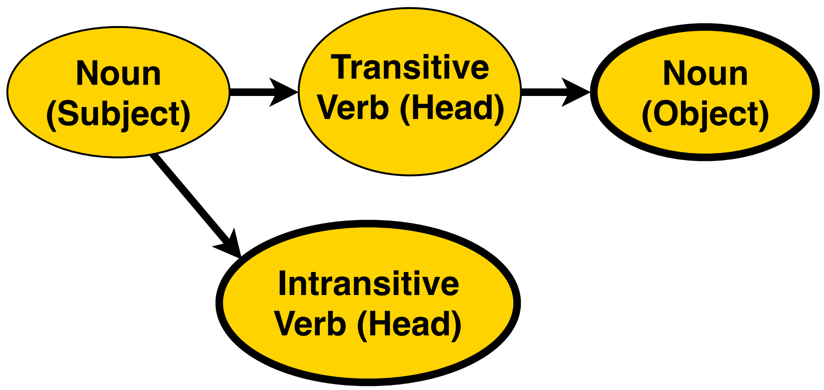
Words take arguments

I eat sushi. ✔
I eat sushi you. ???
I sleep sushi ???
I give sushi ???
I drink sushi ?

Subcategorization
(purely syntactic: what set of arguments do words take?)
Intransitive verbs (sleep) take only a subject.
Transitive verbs (eat) take also one (direct) object.
Ditransitive verbs (give) take also one (indirect) object.

Selectional preferences
(semantic: what types of arguments do words tend to take)
The object of eat should be edible.

A better FSA

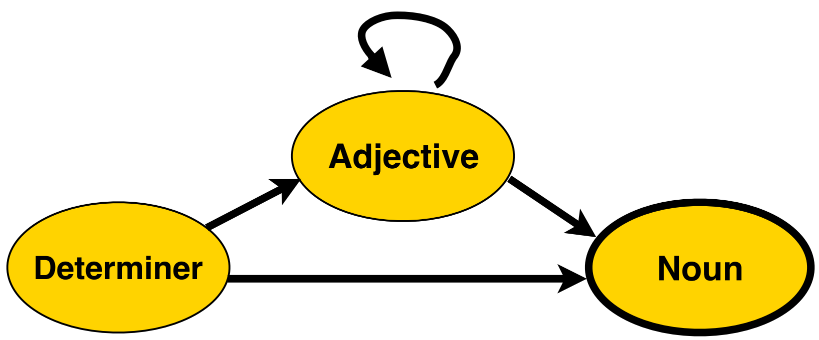


Language is recursive

the ball
the big ball
the big, red ball
the big, red, heavy ball
....

Adjectives can **modify** nouns.
The **number of modifiers (aka adjuncts)** a word can have is (in theory) **unlimited**.

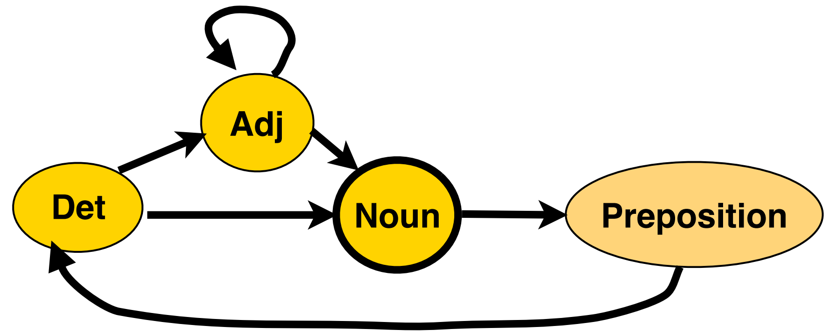
Another FSA



Recursion can be more complex

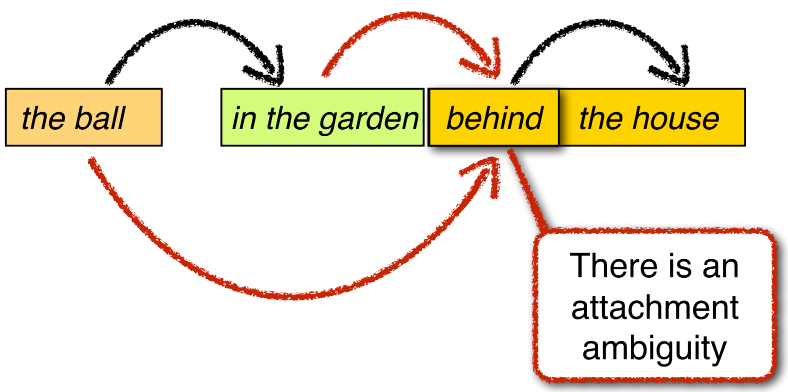
the ball
the ball in the garden
the ball in the garden behind the house
the ball in the garden behind the house next to the school
....

Yet another FSA

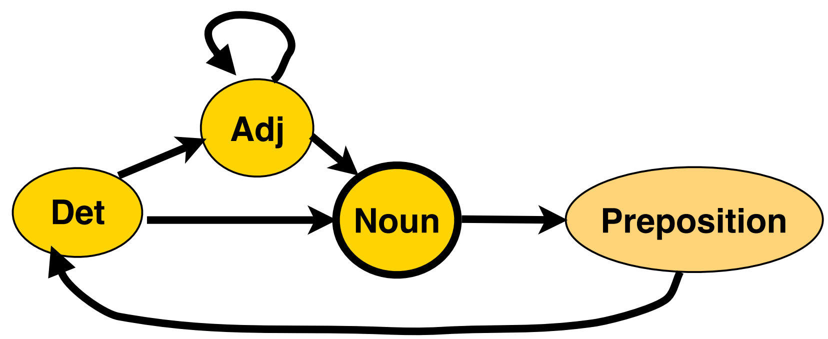


So, why do we need anything beyond regular (finite-state) grammars?

What does this mean?



FSAs do not generate hierarchical structure



What is the structure of a sentence?

Sentence structure is **hierarchical**:

A sentence consists of **words** (I, eat, sushi, with, tuna)
..which form phrases or **constituents**: "sushi with tuna"

Sentence structure defines **dependencies** between words or phrases:

[*I* [*eat* [*sushi* [*with tuna*]]]]

Strong vs. weak generative capacity

Formal language theory:

- defines language as string sets
- is only concerned with generating these strings (*weak* generative capacity)

Formal/Theoretical syntax (in linguistics):

- defines language as sets of strings with (hidden) structure
- is also concerned with generating the right *structures* (*strong* generative capacity)

Context-free grammars (CFGs) capture recursion

Language has complex constituents
("the garden behind the house")

Syntactically, these constituents behave just like simple ones.
("behind the house" can always be omitted)

CFGs define nonterminal categories to capture equivalent constituents.

Context-free grammars

A CFG is a 4-tuple $\langle \mathbf{N}, \mathbf{\Sigma}, \mathbf{R}, S \rangle$ consisting of:

A set of nonterminals $\mathbf{N}$
(e.g. $\mathbf{N}$ = {S, NP, VP, PP, Noun, Verb, ...})

A set of terminals $\mathbf{\Sigma}$
(e.g. $\mathbf{\Sigma}$ = {I, you, he, eat, drink, sushi, ball, })

A set of rules $\mathbf{R}$
$\mathbf{R} \subseteq \{A \rightarrow \beta$ with left-hand-side (LHS) $A \in \mathbf{N}$
and right-hand-side (RHS) $\beta \in (\mathbf{N} \cup \mathbf{\Sigma})^* \}$

A start symbol $S \in \mathbf{N}$

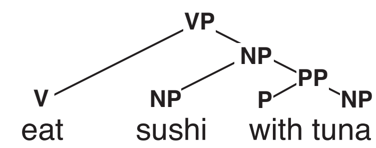
An example

DT → {the, a}
N → {ball, garden, house, sushi }
P → {in, behind, with}
NP → DT N
NP → NP PP
PP → P NP

N: noun
P: preposition
NP: "noun phrase"
PP: "prepositional phrase"

CFGs define parse trees

N → {sushi, tuna}
P → {with}
V → {eat}
NP → N
NP → NP PP
PP → P NP
VP → V NP



CFGs and center embedding

The mouse ate the corn.
The mouse that the snake ate ate the corn.
The mouse that the snake that the hawk ate ate ate the corn.
....

CFGs and center embedding

Formally, these sentences are all grammatical, because they can be generated by the CFG that is required for the first sentence:

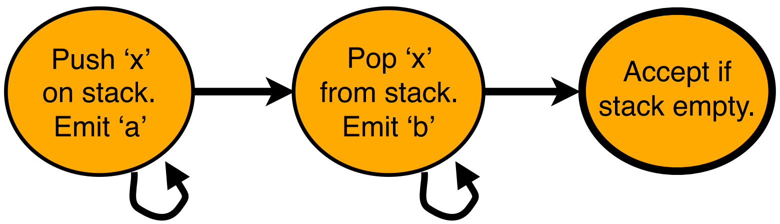
S → NP VP
NP → NP RelClause
RelClause → that NP ate

Problem: CFGs are not able to capture **bounded recursion.** ('only embed one or two relative clauses').

To deal with this discrepancy between what the model predicts to be grammatical, and what humans consider grammatical, linguists distinguish between a speaker's **competence** (grammatical knowledge) and **performance** (processing and memory limitations)

CFGs are equivalent to Pushdown automata (PDAs)

PDAs are FSAs with an additional stack:
Emit a symbol and push/pop a symbol from the stack



This is equivalent to the following CFG:

S → a X b   S → a b
X → a X b   X → a b

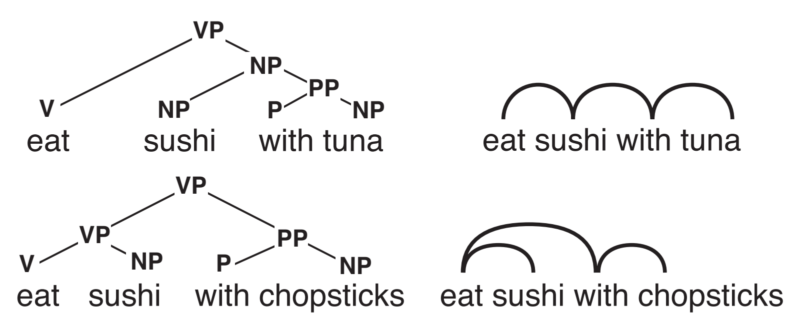
Generating $a^n b^n$

Action	Stack	String
1. Push x on stack. Emit a.	x	a
2. Push x on stack. Emit a.	xx	aa
3. Push x on stack. Emit a.	xxx	aaa
4. Push x on stack. Emit a.	xxxx	aaaa
5. Pop x off stack. Emit b.	xxx	aaaab
6. Pop x off stack. Emit b.	xx	aaaabb
7. Pop x off stack. Emit b.	x	aaaabbb
8. Pop x off stack. Emit b		aaaabbbb

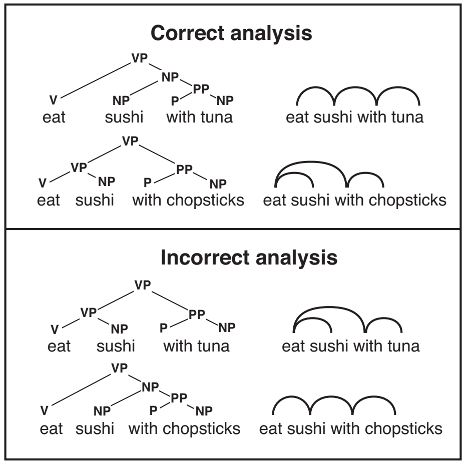
Defining grammars for natural language

Two ways to represent structure

Phrase structure trees **Dependency trees**



Structure (syntax) corresponds to meaning (semantics)



Dependency grammar

DGs describe the structure of sentences as a directed acyclic graph.

The **nodes** of the graph are the **words**
The **edges** of the graph are the **dependencies**.

Typically, the graph is assumed to be a **tree**.

Note: the relationship between DG and CFGs:

If a CFG phrase structure tree is translated into DG, the resulting dependency graph has no crossing edges.

Constituents: Heads and dependents

There are different kinds of constituents:

Noun phrases: the man, a girl with glasses, Illinois
Prepositional phrases: with glasses, in the garden
Verb phrases: eat sushi, sleep, sleep soundly

Every phrase has a **head**:

Noun phrases: the <u>man</u>, a <u>girl</u> with glasses, <u>Illinois</u>
Prepositional phrases: <u>with</u> glasses, <u>in</u> the garden
Verb phrases: <u>eat</u> sushi, <u>sleep</u>, <u>sleep</u> soundly

The other parts are its **dependents**.
Dependents are either **arguments** or **adjuncts**

Is string α a constituent?

He talks [in class].

Substitution test:

Can α be replaced by a single word?
He talks [there].

Movement test:

Can α be moved around in the sentence?
[In class], he talks.

Answer test:

Can α be the answer to a question?
Where does he talk? - [In class].

Arguments are obligatory

Words subcategorize for specific sets of arguments:
Transitive verbs (sbj + obj): [John] likes [Mary]

All arguments have to be present:
*[John] likes. *likes [Mary].

No argument can be occupied multiple times:
*[John] [Peter] likes [Ann] [Mary].

Words can have multiple subcat frames:
Transitive eat (sbj + obj): [John] eats [sushi].
Intransitive eat (sbj): [John] eats.

Adjuncts are optional

Adverbs, PPs and adjectives can be adjuncts:
Adverbs: John runs [fast].
a [very] heavy book.
PPs: John runs [in the gym].
the book [on the table]
Adjectives: a [heavy] book

There can be an arbitrary number of adjuncts:
John saw Mary.
John saw Mary [yesterday].
John saw Mary [yesterday] [in town]
John saw Mary [yesterday] [in town] [during lunch]
[Perhaps] John saw Mary [yesterday] [in town] [during lunch]

A context-free grammar for a fragment of English

Noun phrases (NPs)

Simple NPs:
[He] sleeps. (pronoun)
[John] sleeps. (proper name)
[A student] sleeps. (determiner + noun)

Complex NPs:
[A tall student] sleeps. (det + adj + noun)
[The student in the back] sleeps. (NP + PP)
[The student who likes MTV] sleeps. (NP + Relative Clause)

The NP fragment

NP → Pronoun
NP → ProperName
NP → Det Noun

Det → {a, the, every}
Pronoun → {he, she,...}
ProperName → {John, Mary,...}
Noun → AdjP Noun
Noun → N
NP → NP PP
NP → NP RelClause

Adjective phrases (AdjP) and prepositional phrases (PP)

AdjP → Adj
AdjP → Adv AdjP
Adj → {big, small, red,...}
Adv → {very, really,...}

PP → P NP
P → {with, in, above,...}

The verb phrase (VP)

He [eats].
He [eats sushi].
He [gives John sushi].
He [eats sushi with chopsticks].

VP → V
VP → V NP
VP → V NP PP
VP → VP PP

V → {eats, sleeps gives,...}

Capturing subcategorization

He [eats]. ✔
He [eats sushi]. ✔
He [gives John sushi]. ✔
He [eats sushi with chopsticks]. ✔
*He [eats John sushi]. ???

VP → $V_{intrans}$
VP → V_{trans} NP
VP → $V_{ditrans}$ NP NP
VP → VP PP
$V_{intrans}$ → {eats, sleeps}
V_{trans} → {eats}
V_{trans} → {gives}

Sentences

[He eats sushi].
[Sometimes, he eats sushi].
[In Japan, he eats sushi].

S → NP VP
S → AdvP S
S → PP S

He says [he eats sushi].
VP → Vcomp S
Vcomp → {says, think, believes}

Sentences redefined

[He eats sushi]. ✔
*[I eats sushi]. ???
*[They eats sushi]. ???

S → NP_{3sg} VP_{3sg}
S → NP_{1sg} VP_{1sg}
S → NP_{3pl} VP_{3pl}

We need features to capture agreement:
(number, person, case,…)

Complex VPs

In English, simple tenses have separate forms:

present tense: the girl eats sushi
simple past tense: the girl ate sushi

Complex tenses, progressive aspect and passive voice consist of auxiliaries and participles:

past perfect tense: the girl has eaten sushi
future perfect: the girl will have eaten sushi
passive voice: the sushi was eaten by the girl
progressive: the girl is/was/will be eating sushi

VPs redefined

He [has [eaten sushi]].
The sushi [was [eaten by him]].

VP → V_{have} $VP_{pastPart}$
VP → V_{be} VP_{pass}
$VP_{pastPart}$ → $V_{pastPart}$ NP
VP_{pass} → $V_{pastPart}$ PP
V_{have} → {has}
$V_{pastPart}$ → {eaten, seen}

We need more nonterminals (e.g. $VP_{pastpart}$).
N.B.: We call $VP_{pastPart}$, VP_{pass}, etc. `untensed' VPs

Coordination

[He eats sushi] and [she drinks tea]
[John] and [Mary] eat sushi.
He [eats sushi] and [drinks tea]

$S \rightarrow S$ conj S
$NP \rightarrow NP$ conj NP
$VP \rightarrow VP$ conj VP

He says [he eats sushi].
$VP \rightarrow V_{comp}\ S$
$V_{comp} \rightarrow \{$says, think, believes$\}$

Relative clauses

Relative clauses modify a noun phrase:
the girl [that eats sushi]

Relative clauses lack a noun phrase, which is understood to be filled by the NP they modify:
'the girl that eats sushi' implies 'the girl eats sushi'

There are subject and object relative clauses:
subject: 'the girl that eats sushi'
object: 'the sushi that the girl eats'

Yes/No questions

Yes/no questions consist of an auxiliary, a subject and an (untensed) verb phrase:

does she eat sushi?
have you eaten sushi?

$\text{YesNoQ} \rightarrow \text{Aux NP VP}_{inf}$
$\text{YesNoQ} \rightarrow \text{Aux NP VP}_{pastPart}$

Wh-questions

Subject wh-questions consist of an wh-word, an auxiliary and an (untensed) verb phrase:

Who has eaten the sushi?

Object wh-questions consist of an wh-word, an auxiliary, an NP and an (untensed) verb phrase:

What does Mary eat?

Today's key concepts

Natural language syntax

- Constituents
- Dependencies
- Context-free grammar
- Arguments and modifiers
- Recursion in natural language

Today's reading

Textbook:

- Jurafsky and Martin, Chapter 12, sections 1-7