ANLP

14 - CFGs, CKY (structure, part II)

David Schlangen
University of Potsdam, MSc Cognitive Systems
Winter 2019 / 2020

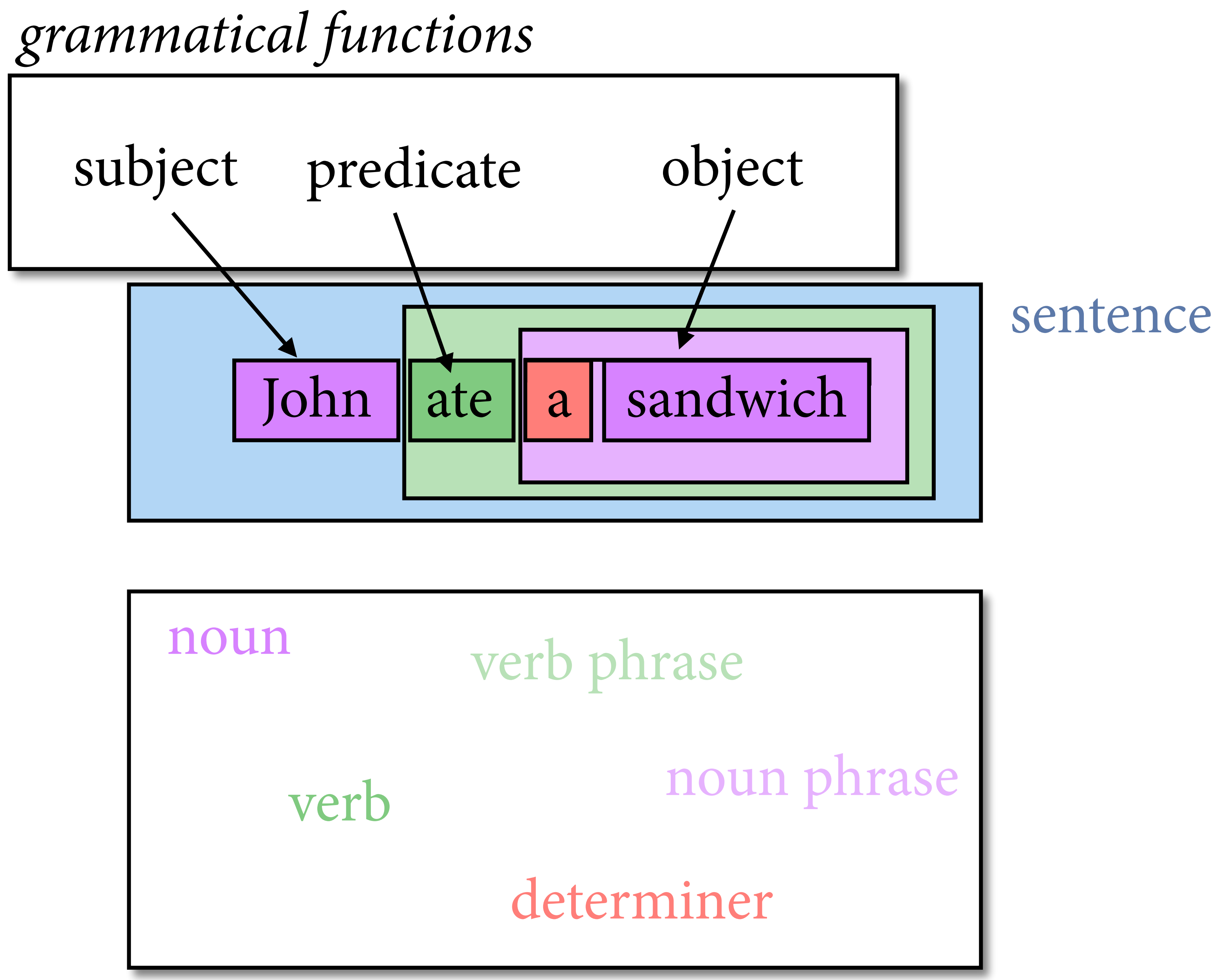
where are we?

- words, and how to represent them
- sequences of words, modelled with n-gram models (generative model)
- classification of objects, with small label set; weighting of features of object (generative classifier: Naive Bayes; discriminative classifiers: logistic regression, SVMs)
- classification of structured objects, with structured labels (generative: HMMs, discriminative: CRFs)
- yet more methods: NNs, forward and recurrent; pre-training. Gives us continuous representations (= vectors) of words (in context) and sequences.
- today: explicit structural representations of sequences

syntax

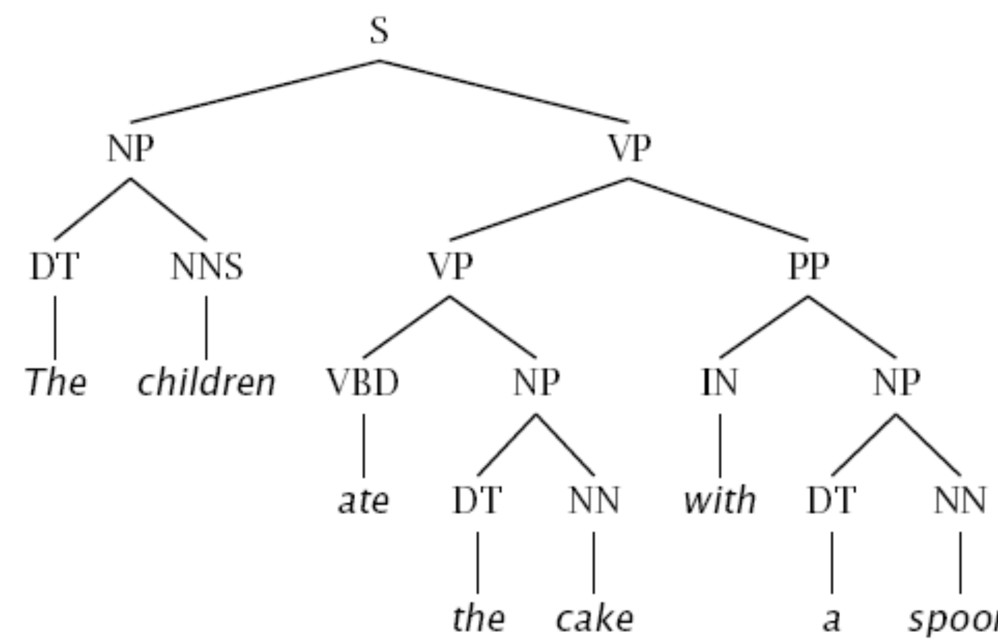
- let's assume we want to know who did what to whom
- POS-tags are not enough:
  - *I ate the spaghetti with chopsticks*
  - *I ate the spaghetti with meatballs*
  - *PP VBD DT NN IN NNS*
- We need more structure that tells us about relations btw parts of sentence.
- first: *all* structures; later: best (= ?) structure

Sentences have structure



today: constituent structure

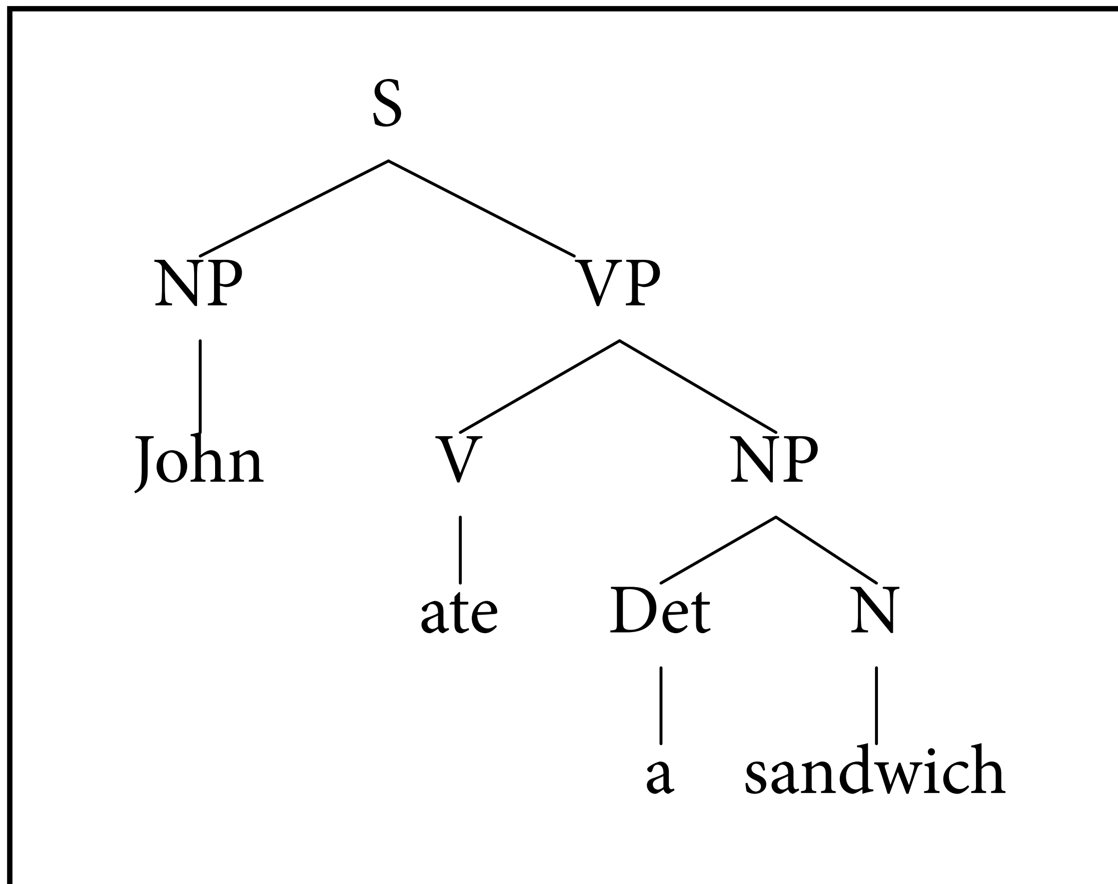
- How do we know what the constituents are?
- Constituency tests:
  - Substitution by *proform* (e.g., pronoun)
  - Clefting (*It was with a spoon that…*)
  - Answer ellipsis (What did they eat? *the cake*)
    (How? *with a spoon*)



- Sometimes constituency is not clear, e.g., coordination:
  she went to and bought food at the store

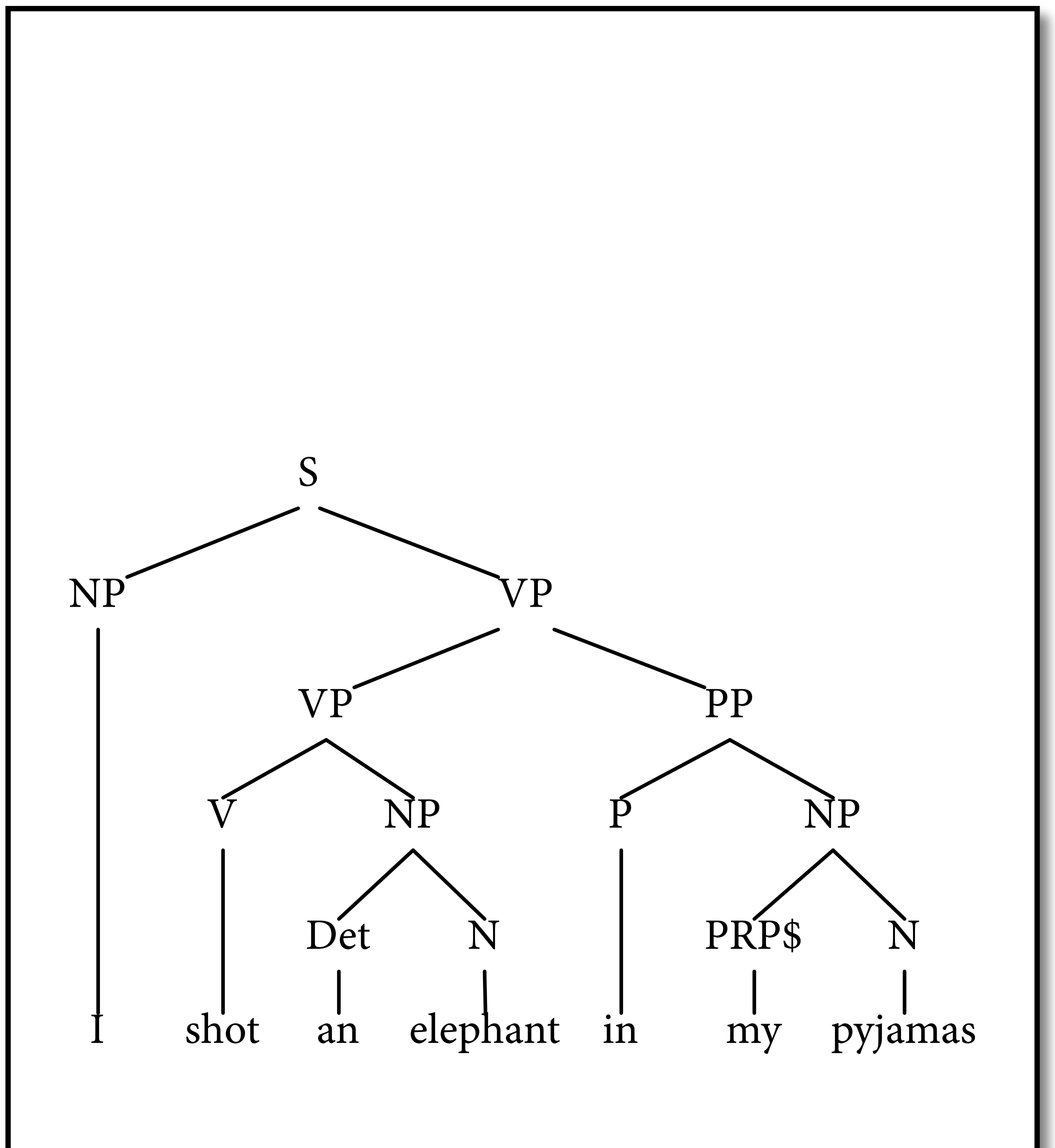
Sentences have structure

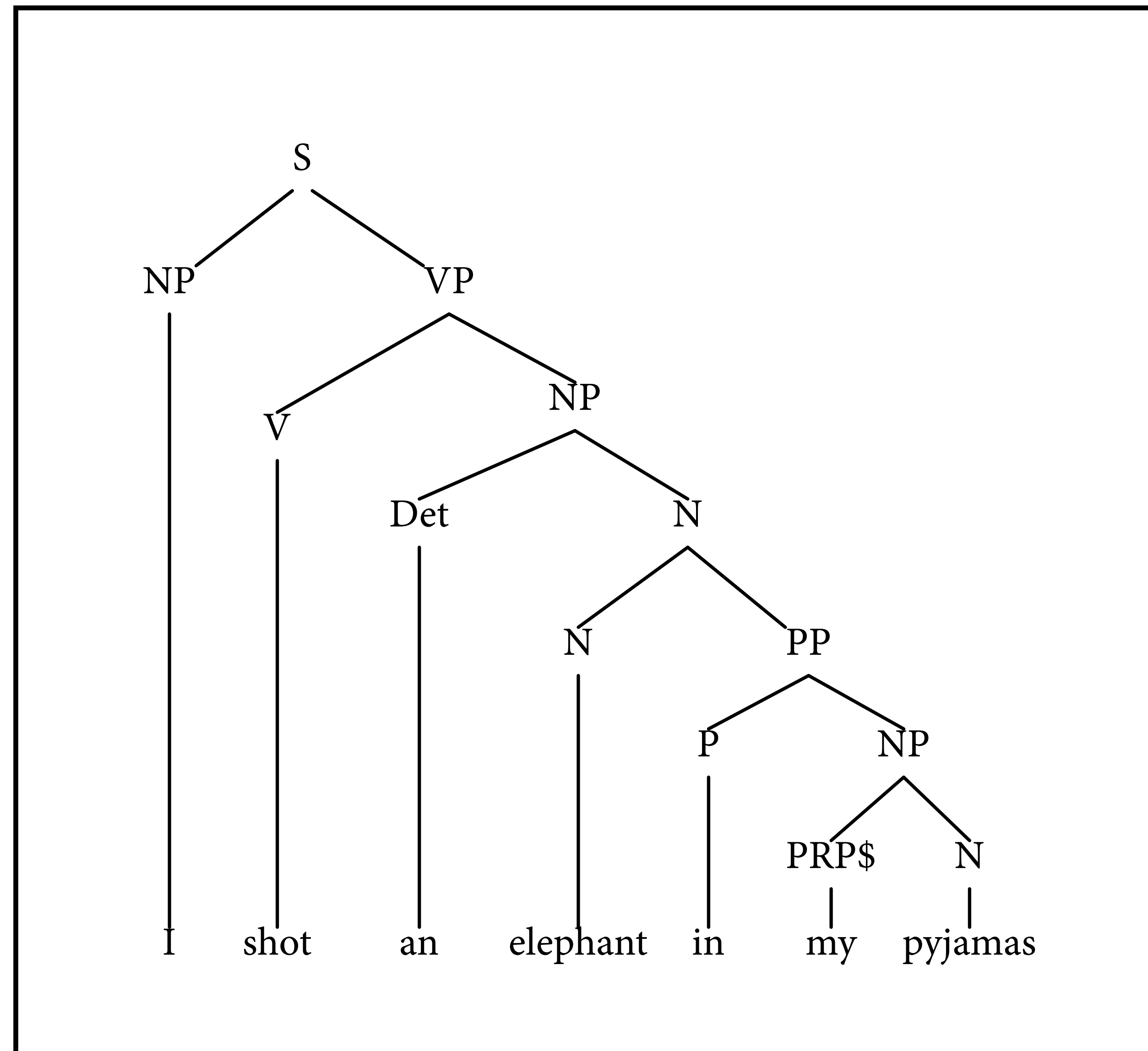
Record it conveniently in *phrase structure tree.*



Ambiguity

Special challenge: sentences can have many possible structures.





This sentence is example of *attachment ambiguity.*

Grammars

- A *grammar* is a finite device for describing large (possibly infinite) set of strings.
  - strings = NL expressions of various types
  - grammar captures linguistic knowledge about syntactic structure

- There are many different grammar formalisms that are being used in NLP.

- In this course we focus on *context-free grammars*.

Context-free grammars

- Context-free grammar (cfg) G is 4-tuple (N,T,S,P):
  - N and T are disjoint finite sets of symbols:
    T = *terminal* symbols; N = *nonterminal* symbols.
  - $S \in N$ is the *start symbol.*
  - P is a finite set of *production rules* of the form $A \rightarrow w$,
    where A is nonterminal and w is a string from $(N \cup T)^*$.

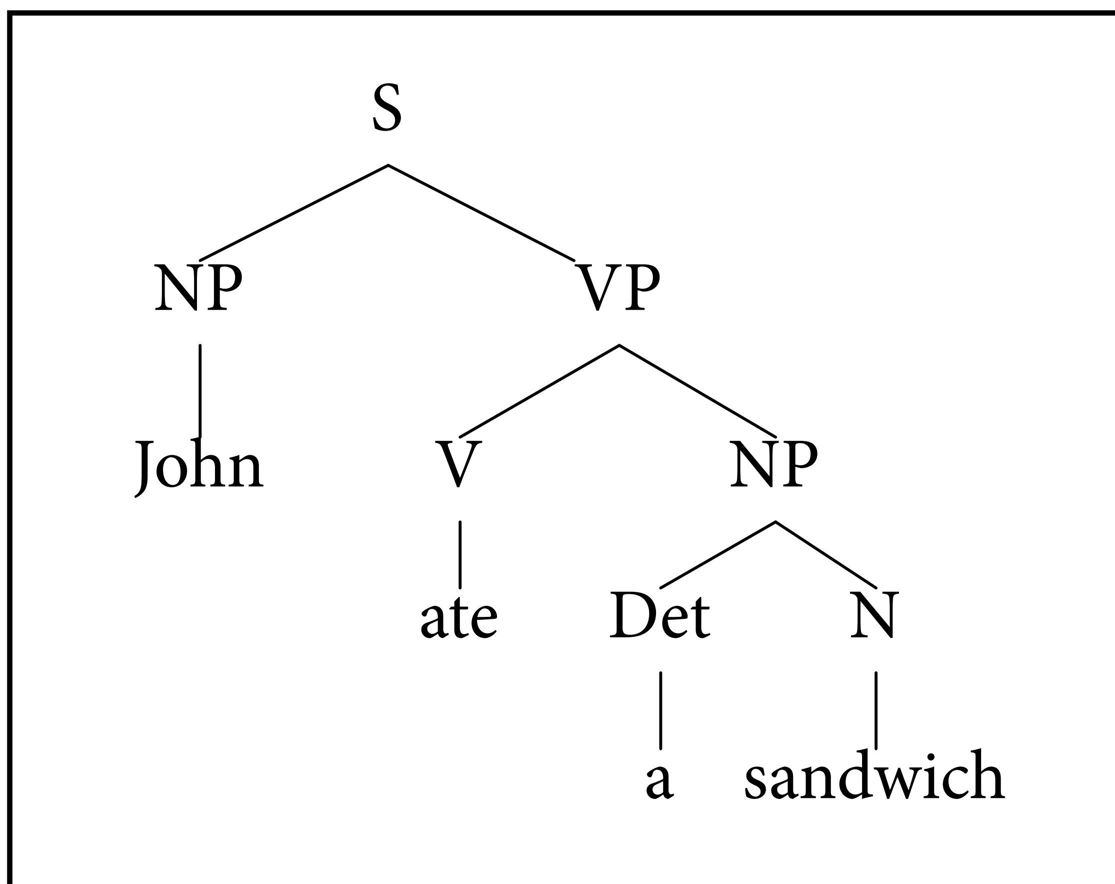
- Why "context-free"?
  - Left-hand side of production is a single nonterminal A.
  - Rule can't look at context in which A appears.
  - *Context-sensitive* grammars can do that.

Example

T = {John, ate, sandwich, a}
N = {S, NP, VP, V, N, Det}; start symbol: S

Production rules:

S → NP VP
NP → Det N
VP → V NP
V → ate
NP → John
Det → a
N → sandwich



perspectives on grammar

- device for characterising set (sentences of language / the language itself)
- „classifier“: grammatical yes / no
- „classifier“: assigns complex label (tree) to string (or FAIL)
- generative device: generates all sentences of language (eventually)
- today: formal device that can be used in algorithm, to analyse input string

Some important concepts

- *One-step derivation* relation $\Rightarrow$:

  $w_1 \; A \; w_2 \Rightarrow w_1 \; w \; w_2$ iff $A \rightarrow w$ is in P

  (w_1, w_2, w are strings from $(N \cup T)^*$)

- *Derivation* relation $\Rightarrow^*$ is reflexive, transitive closure:

  $w \Rightarrow^* w_n$ if $w \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$ (for some $n \geq 0$)

- *Language* $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$

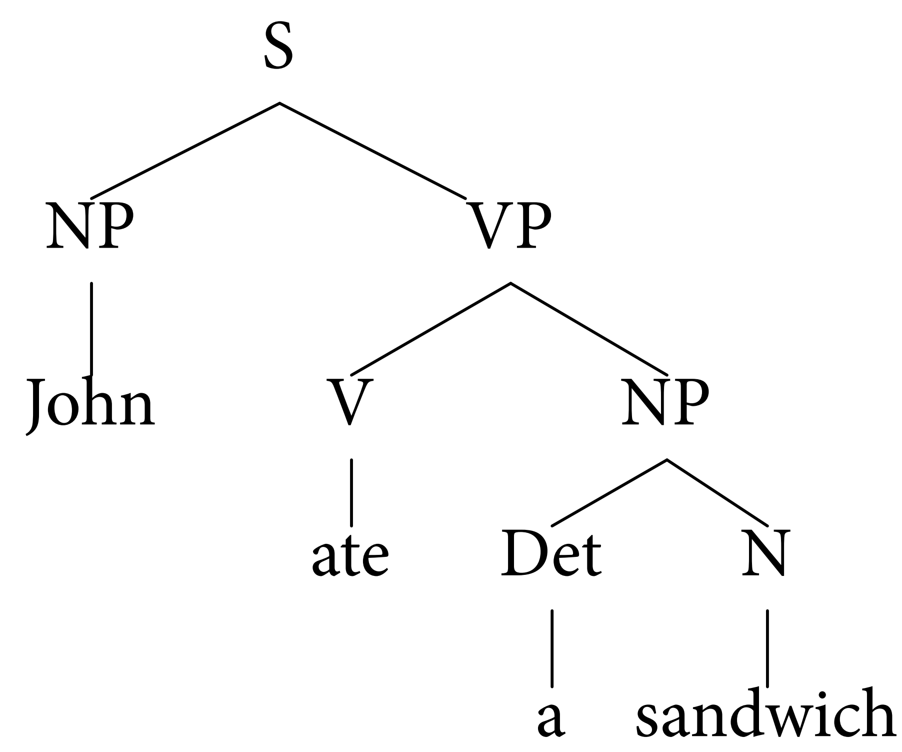
Derivations and parse trees

Parse tree provides readable, high-level view of derivation.

derivation

$$
\begin{aligned}
S &\Rightarrow \text{NP VP} \Rightarrow \text{John VP} \\
&\Rightarrow \text{John V NP} \Rightarrow \text{John ate NP} \\
&\Rightarrow \text{John ate Det N} \\
&\Rightarrow \text{John ate a N} \\
&\Rightarrow \text{John ate a sandwich}
\end{aligned}
$$

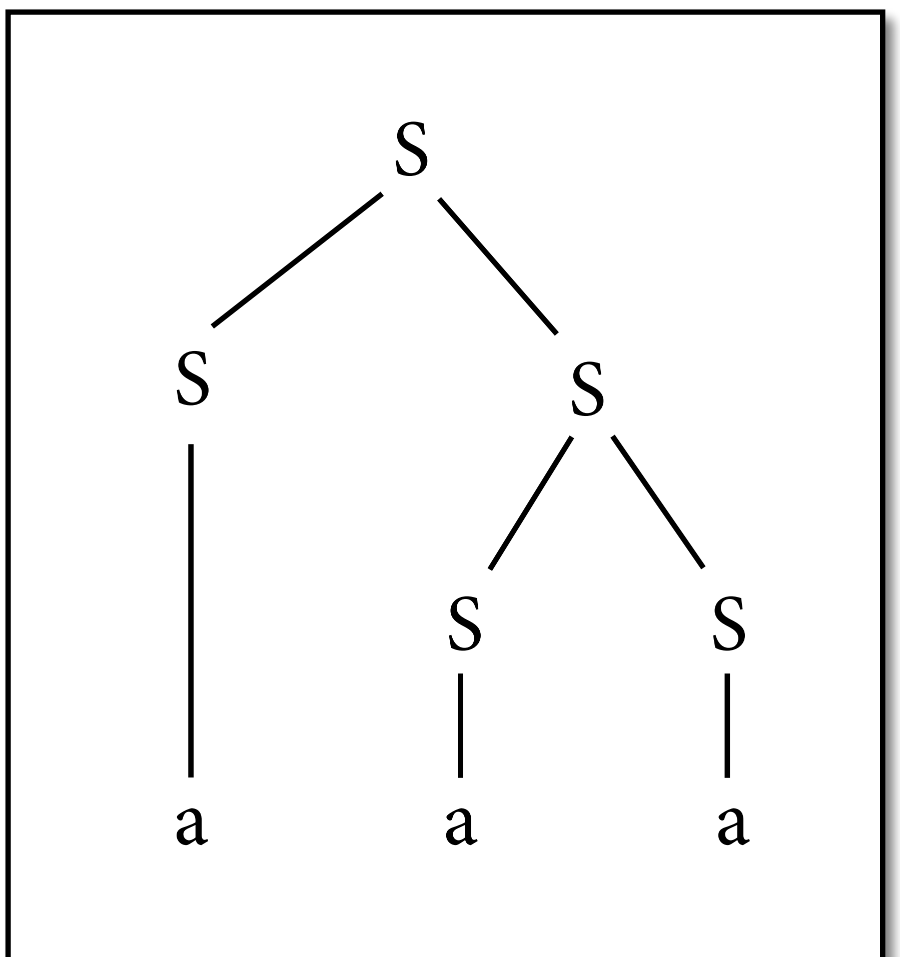
parse tree

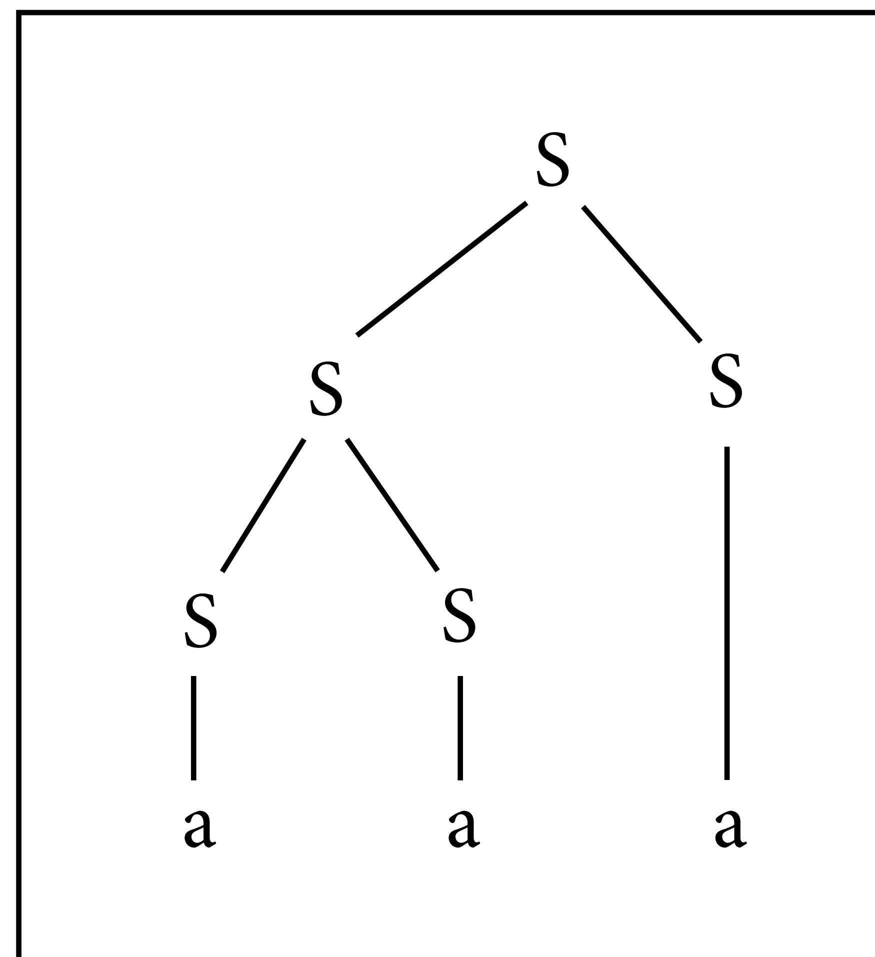


Big languages

Number of parse trees can grow exponentially in string length.

$S \rightarrow S\ S$ $\qquad\qquad$ $S \rightarrow a$





Recognition and parsing

- Let G be a cfg and w be a string.
- *Word problem:* is $w \in L(G)$?
  - Algorithms that solve it are called *recognizers.*
- *Parsing problem:* enumerate all parse trees of w.
  - Algorithms that solve it are called *parsers.*
- Every parser also solves the word problem.

Parsing algorithms

- How can we solve the word and parsing problem so systematically that we can implement it?
- One simple approach: shift-reduce algorithm (here: only for the word problem).
- Then: Analyze efficiency of SR and replace it with faster algorithm: CKY.

demo

```
In [1]: import nltk

In [2]: nltk.app.srparser()
```

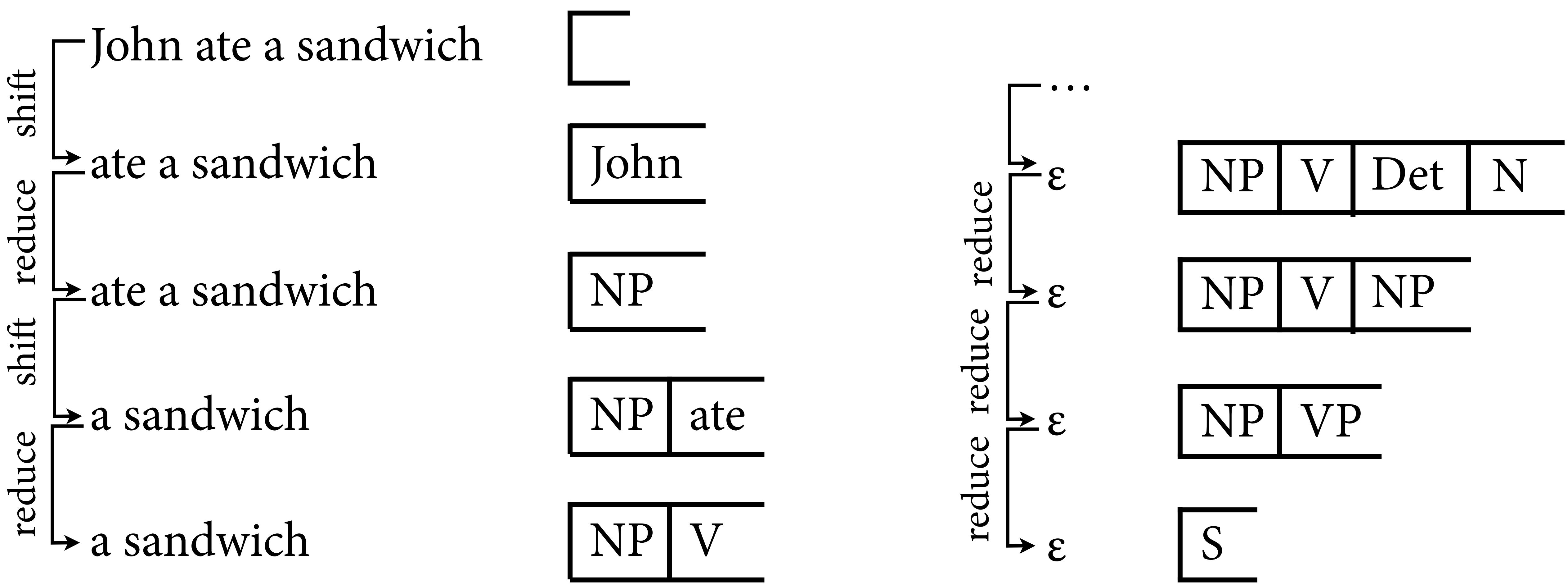
Try to get to a complete parse (a tree spanning the whole input) by repeated applications of the „shift“ and the „reduce“ operation.

Shift-Reduce Parsing

T = {John, ate, sandwich, a}
N = {S, NP, VP, V, N, Det}; start symbol: S

Production rules:

S → NP VP
VP → V NP
V → ate
Det → a
NP → Det N
NP → John
N → sandwich



Shift-Reduce Parsing

- Read input string step by step. In each step, we have
  - the remaining input words we have not shifted yet
  - a *stack* of terminal and nonterminal symbols

- In each step, apply a rule:
  - Shift: moves the next input word to the top of the stack
  - Reduce: applies a production rule to replace top of stack with the nonterminal on the left-hand side

- Sentence is in language of cfg iff we can read the whole string and stack contains only start symbol.

Shift-Reduce Parsing

- Shift rule:
  $(s, a \cdot w) \rightarrow (s \cdot a, w)$

- Reduce rule:
  $(s \cdot w', w) \rightarrow (s \cdot A, w)$ if $A \rightarrow w'$ in P

- Start: (ε, w)

- Apply rules *nondeterministically:*
  Claim $w \in L(G)$ if there *exists* some sequence of steps that derive (S, ε) from (ε, w).

Nondeterminism

- Claim that string is in language of cfg iff (S, ε) can be derived by *any one* sequence of shift and reduce steps.
- This is very important because there are many stack-string pairs where multiple rules can be applied:
  - shift-reduce conflict
  - reduce-reduce conflict
- In practice, we need to try all sequences out.
  - Compilers for programming languages avoid this by careful language design: no ambiguity in grammar.

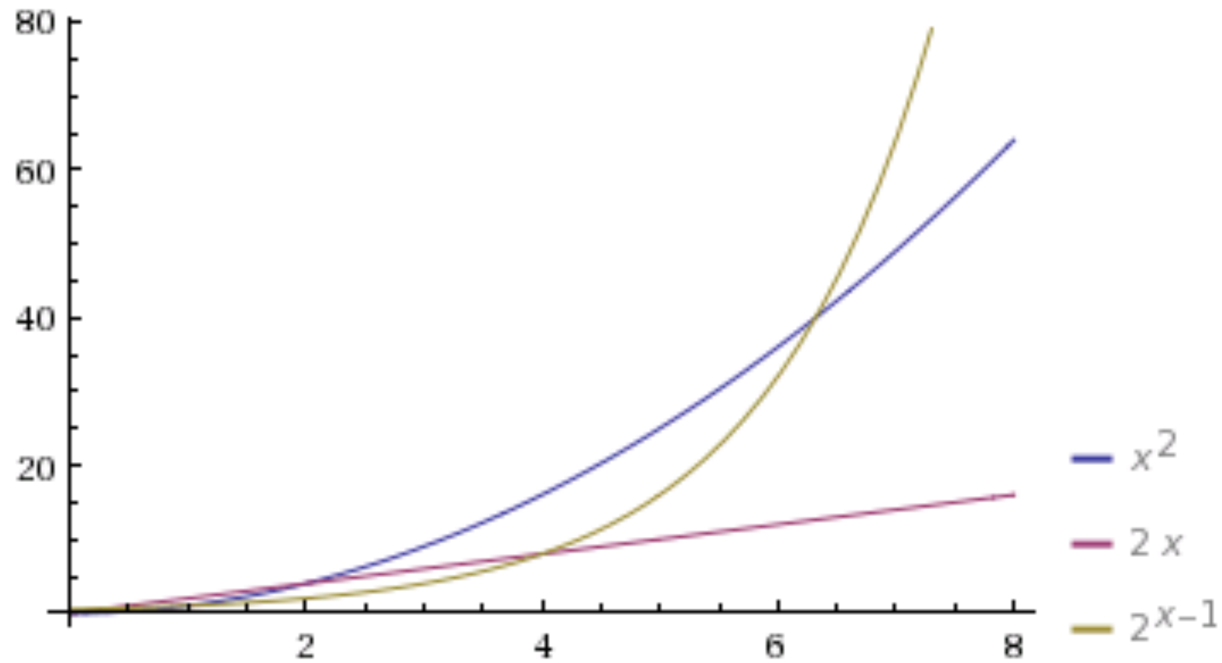
Analyzing Shift-Reduce

- If string has length n and grammar has k nonterminals, then there are $O(k^n)$ ways of assigning strings of nonterminals to words.

- These can all be explored, especially when the string is *not* in the language.

- Big O Notation
- Complexity of algorithm
- Behaviour as function of input size can be described as this function (here: exponential)
- Bad news!

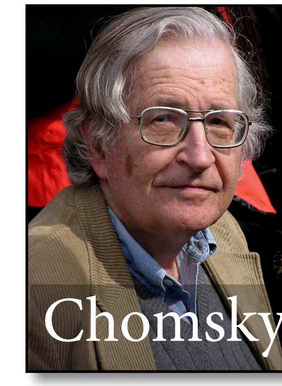
Polynomial vs. exponential



- We often distinguish between *polynomial* and *exponential* runtime. Rule of thumb: exponential = too slow for practical use.

- Is there a polynomial algorithm for the word problem?

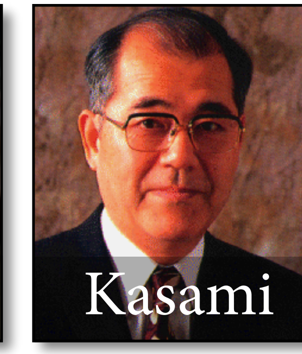
Chomsky Normal Form



- A cfg is *in Chomsky normal form (CNF)* if each of its production rules has one of these two forms:
  - A → B C: right-hand side is exactly two nonterminals
  - A → c: right-hand side is exactly one terminal

- For every cfg G, there is a weakly equivalent cfg G' which is in CNF.
  - that is, L(G) = L(G')

The CKY Algorithm



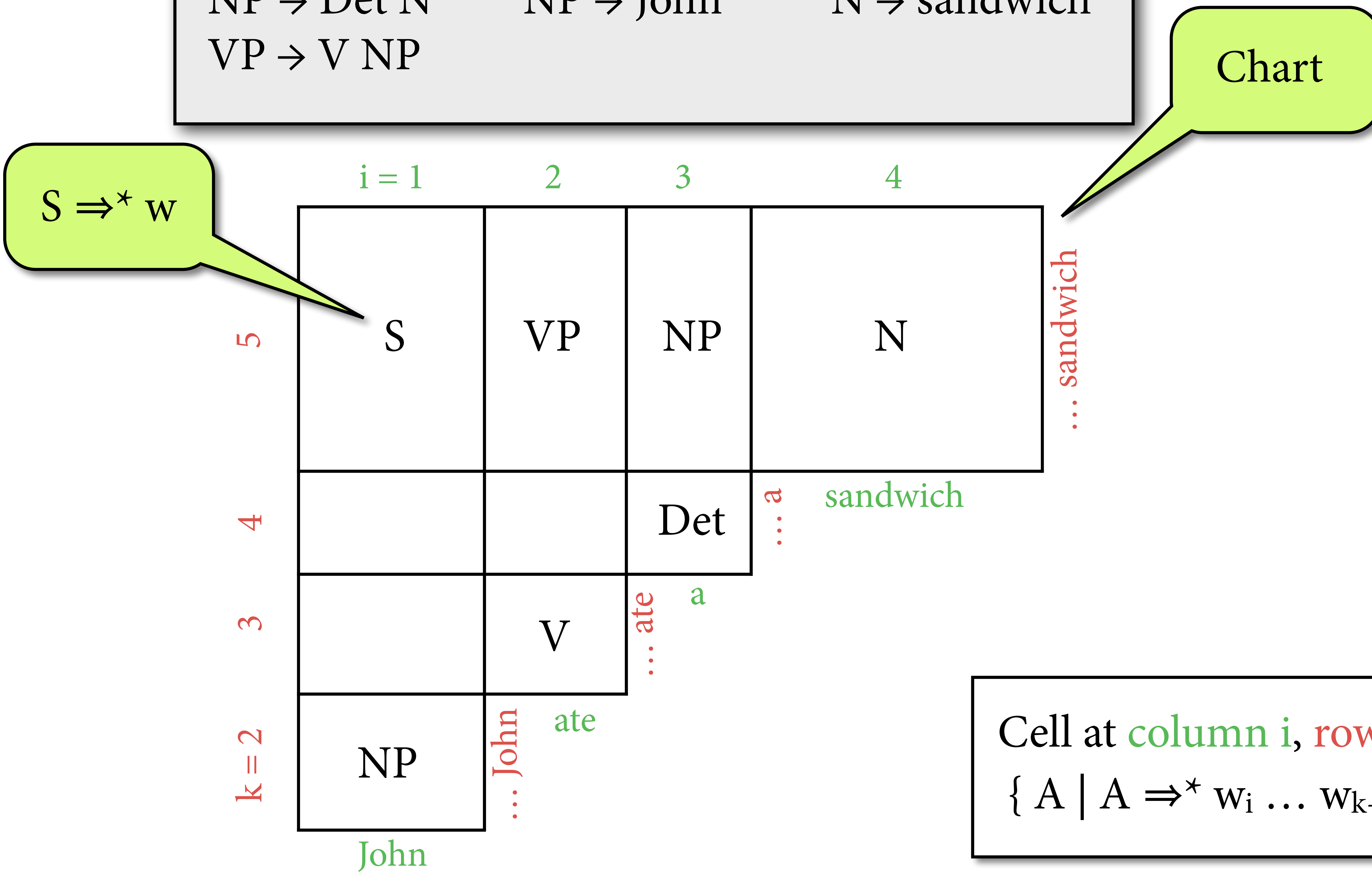
- Simplest and most-used chart parser for cfgs in CNF.

- Developed independently in the 1960s by John Cocke, Daniel Younger, and Tadao Kasami.
  - sometimes also called CYK algorithm

- Bottom-up algorithm for discovering statements of the form "$A \Rightarrow^* w_i \ldots w_{k-1}$?"

The CKY Recognizer

S → NP VP	V → ate	Det → a
NP → Det N	NP → John	N → sandwich
VP → V NP		

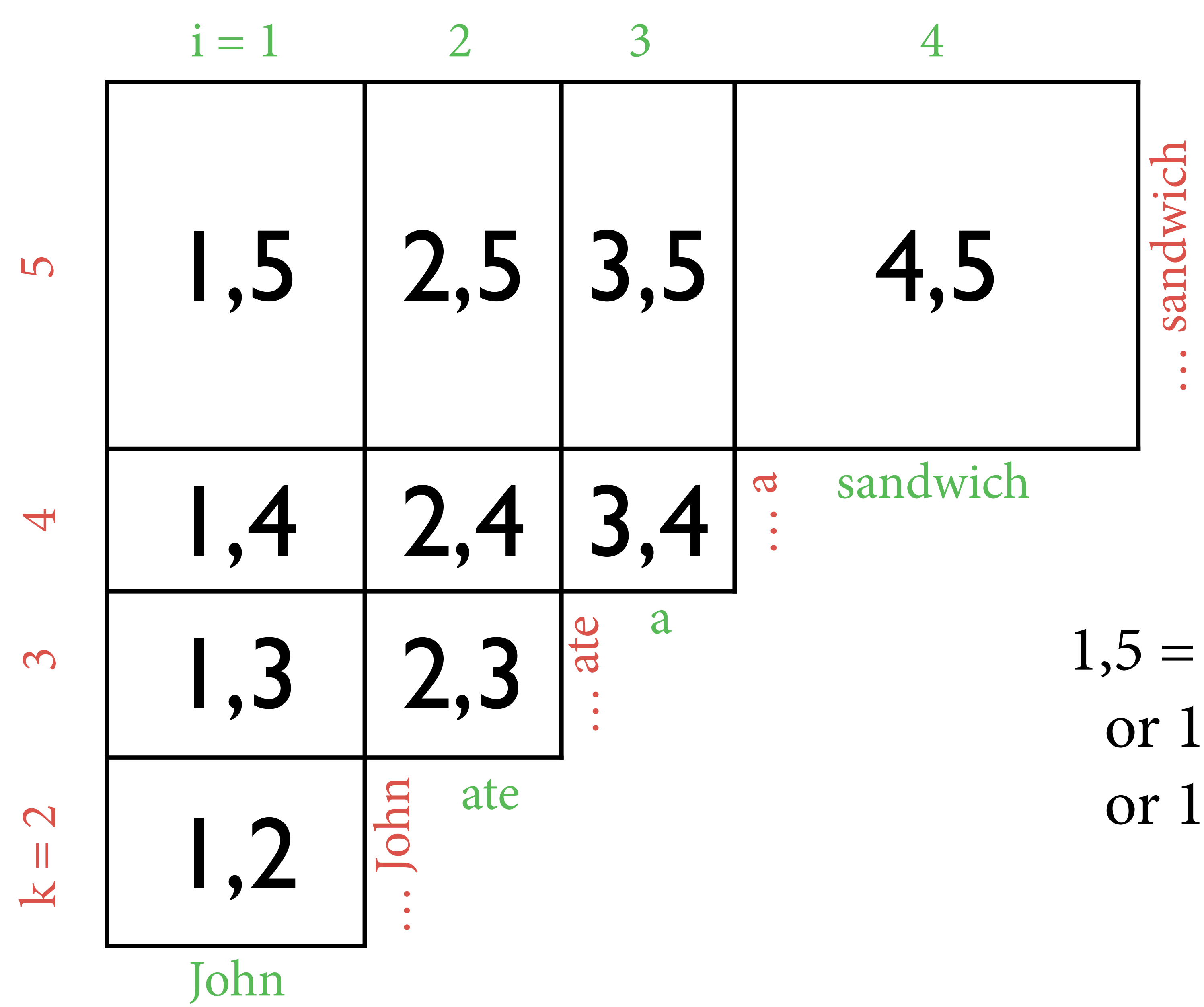


Cell at column i, row k:

$\{ A \mid A \Rightarrow^* w_i \ldots w_{k-1} \}$

The CKY Recognizer

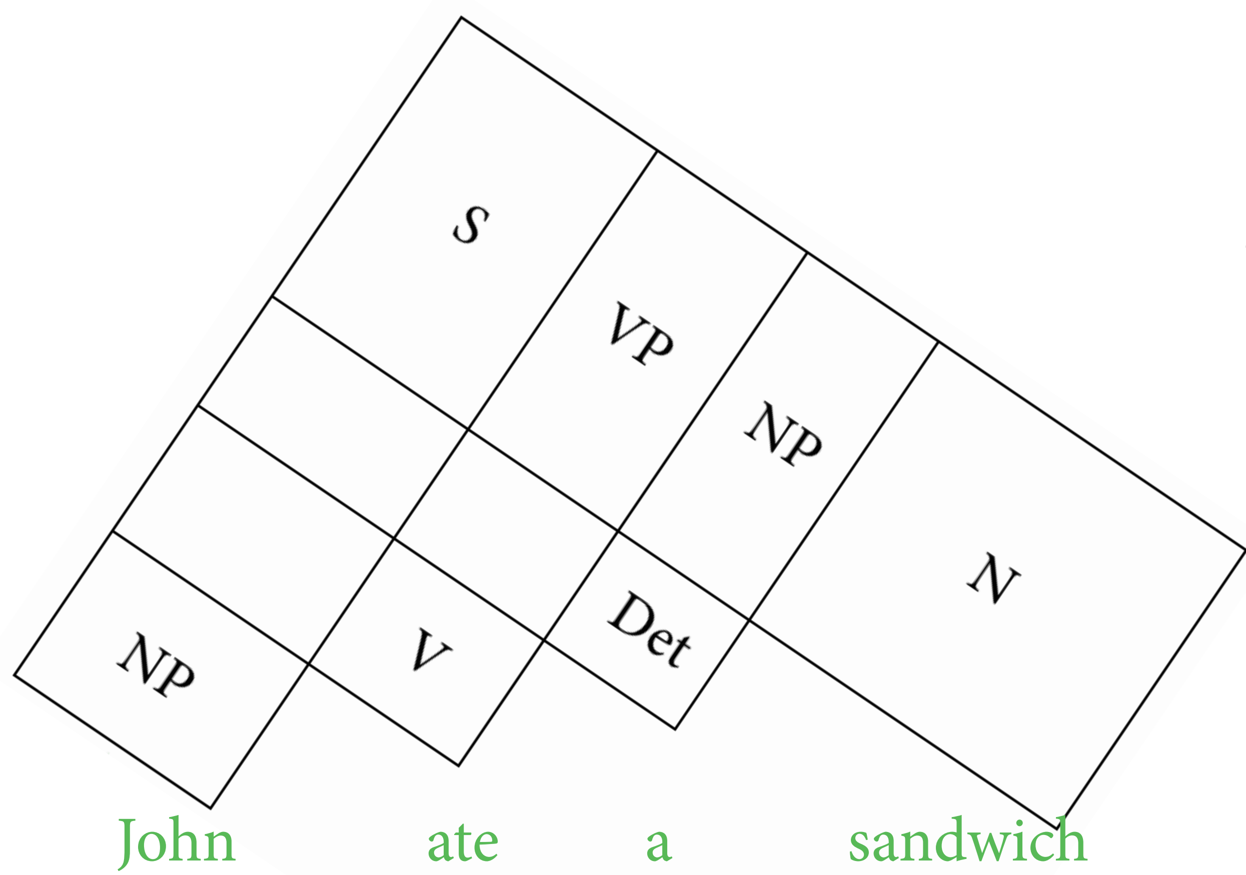
S → NP VP	V → ate	Det → a
NP → Det N	NP → John	N → sandwich
VP → V NP		



1,5 = 1,2 + 2,5
or 1,3 + 3,5
or 1,4 + 4,5

The CKY Recognizer

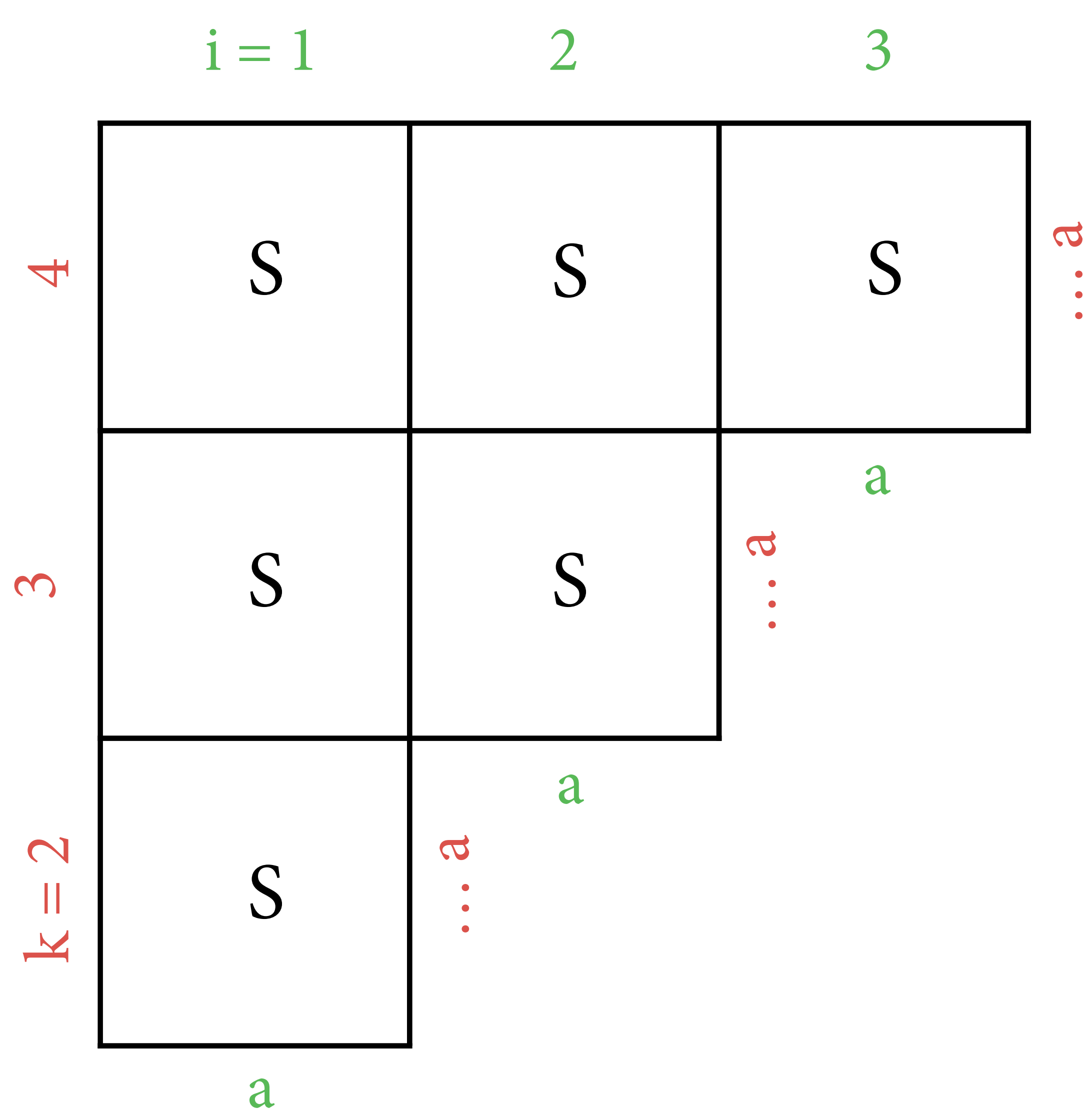
S → NP VP	V → ate	Det → a
NP → Det N	NP → John	N → sandwich
VP → V NP		



perhaps easier to see the trees this way…

The CKY Recognizer

S → S S   S → a



CKY recognizer: pseudocode

Data structure: Ch(i,k) eventually contains $\{A \mid A \Rightarrow^* w_i \ldots w_{k-1}\}$
(initially all empty).

for each i from 1 to n:
    for each production rule $A \rightarrow w_i$:
        add A to Ch(i, i+1)

for each *width* b from 2 to n:
    for each *start position* i from 1 to n-b+1:
        for each *left width* k from 1 to b-1:
            for each $B \in$ Ch(i, i+k) and $C \in$ Ch(i+k,i+b):
                for each production rule $A \rightarrow B\ C$:
                    add A to Ch(i,i+b)

claim that $w \in L(G)$ iff $S \in$ Ch(1,n+1)

Complexity

- *Time* complexity of CKY recognizer is $O(n^3)$, although number of parse trees grows exponentially.
- *Space* complexity of CKY recognizer is $O(n^2)$ (one cell for each substring).
- Efficiency depends crucially on CNF. Naive generalization of CKY to rules $A \rightarrow B_1 \ldots B_r$ raises time complexity to $O(n^{r+1})$.

Correctness

- Soundness: CKY *only* derives true statements.
  - If CKY puts A into Ch(i,k), then there is rule $A \rightarrow BC$ and some j with $B \in Ch(i,j)$ and $C \in Ch(j,k)$.
  - Induction hypothesis: for shorter spans, have $B \Rightarrow^* w_i \dots w_{j-1}$.

    Thus $A \Rightarrow B\ C \Rightarrow^* w_i \dots w_{j-1}\ C \Rightarrow^* w_i \dots w_{k-1}$
- Completeness: CKY derives *all* true statements.
  - Each derivation $A \Rightarrow^* w_i \dots w_{k-1}$ starts with a first step;

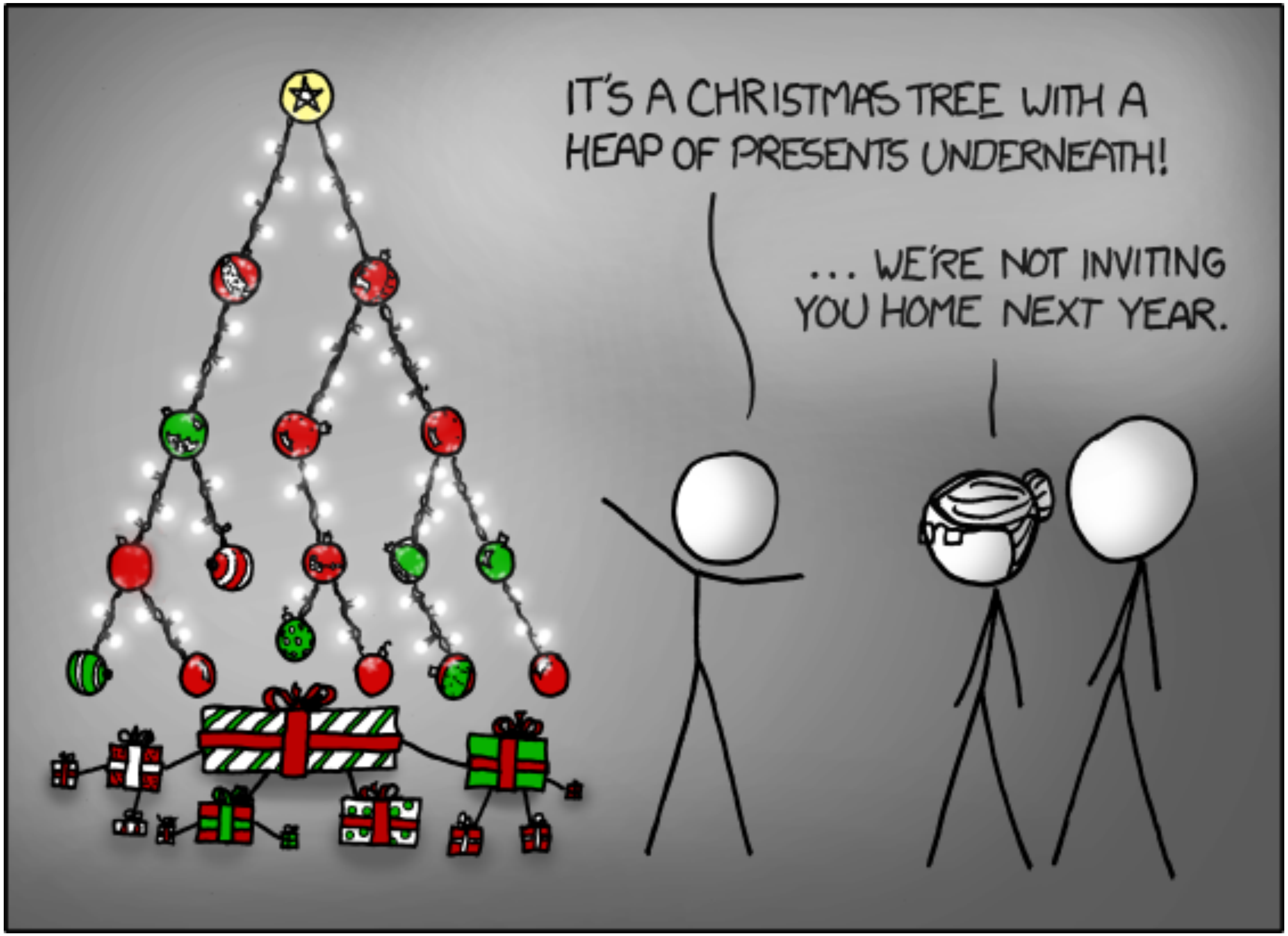
    say $A \Rightarrow B\ C \Rightarrow^* w_i \dots w_{j-1}\ C \Rightarrow^* w_i \dots w_{k-1}$
  - Important: ensure that all nonterminals for shorter spans are known before filling Ch(i,k).

Recognizer to Parser

- Parser: need to construct parse trees from chart.
- Do this by memorizing how each $A \in Ch(i,k)$ can be constructed from smaller parts.
  - built from $B \in Ch(i,j)$ and $C \in Ch(j,k)$ using $A \rightarrow B\ C$: store (B,C,j) in *backpointer* for A in $Ch(i,k)$.
  - analogous to backpointers in HMMs
- Once chart has been filled, enumerate trees recursively by following backpointers, starting at $S \in Ch(1,n+1)$.

Conclusion

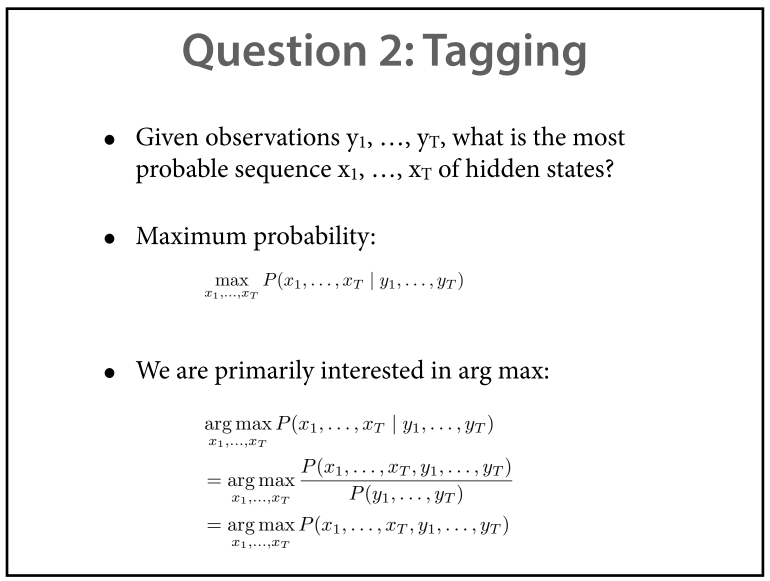
- Context-free grammars: most popular grammar formalism in NLP.
  - there are also other, more expressive grammar formalisms
- CKY: most popular parser for cfgs.
  - very simple polynomial algorithm, works well in practice
  - there are also other, more complicated algorithms
- Next time: put parsing and statistics together.



https://xkcd.com/835/

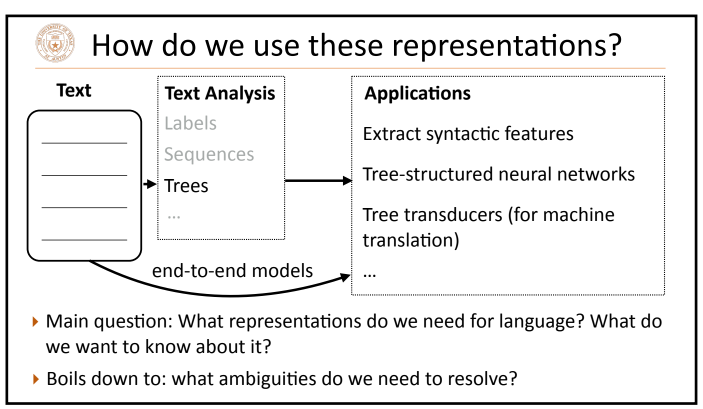
slide credits

slides that look like this



come from

earlier editions of this class (ANLP), given by Alexander Koller



CS388 given by Greg Durrett at U Texas, Austin

and their use is gratefully acknowledged. I try to make any modifications obvious, but if there are errors on a slide, assume that I added them.