Context-free Grammars

Computational Linguistics

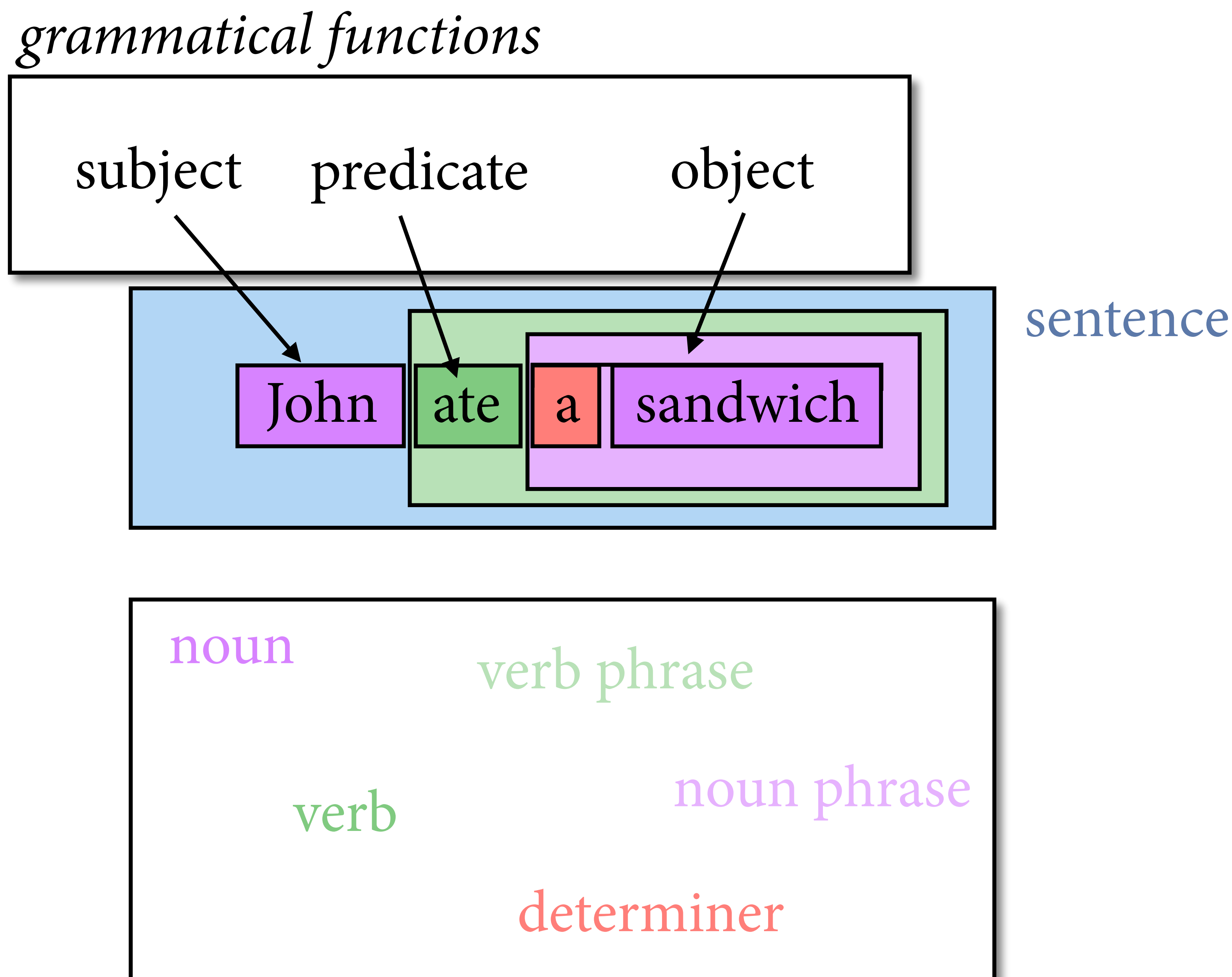
Alexander Koller

14 November 2023

Outline

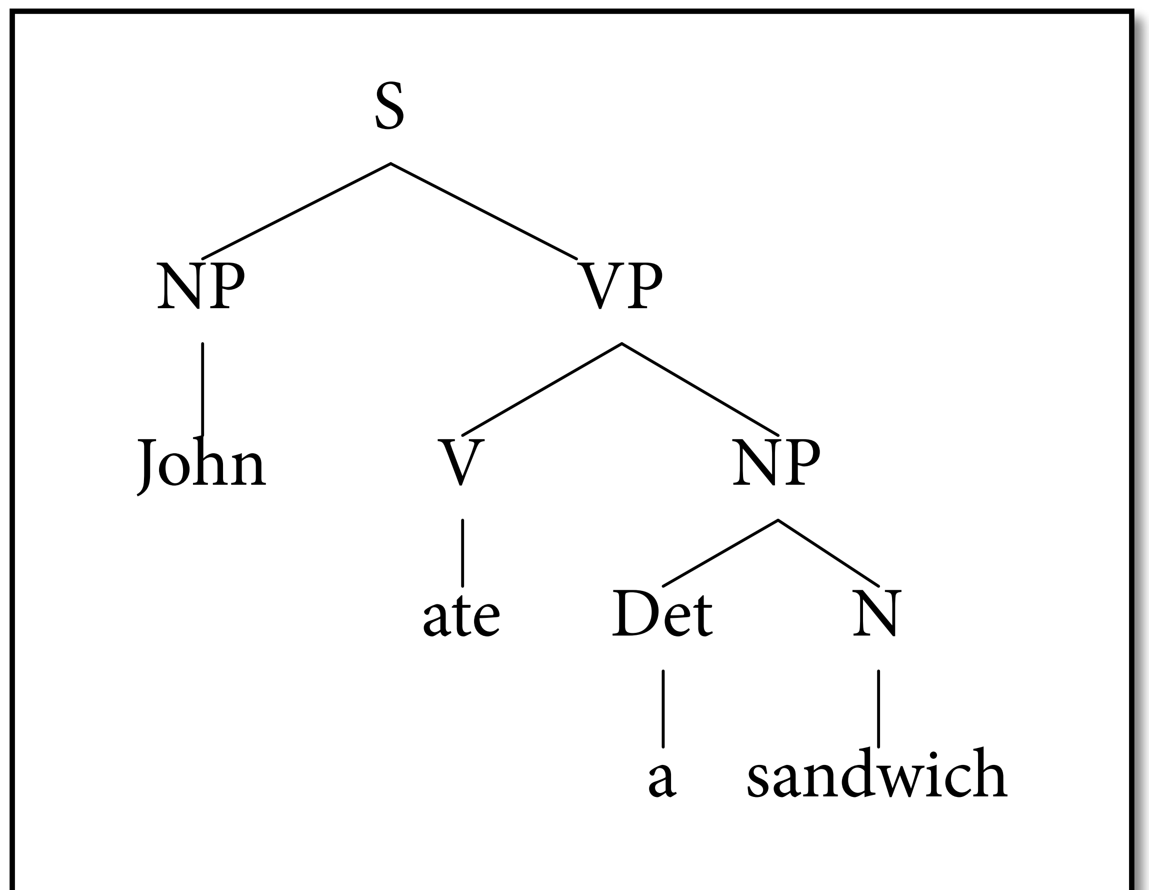
1. Context-free grammars
2. The shift-reduce parser
3. Shift-reduce parsing as a parsing schema
4. Correctness proof of the shift-reduce parser

Sentences have structure



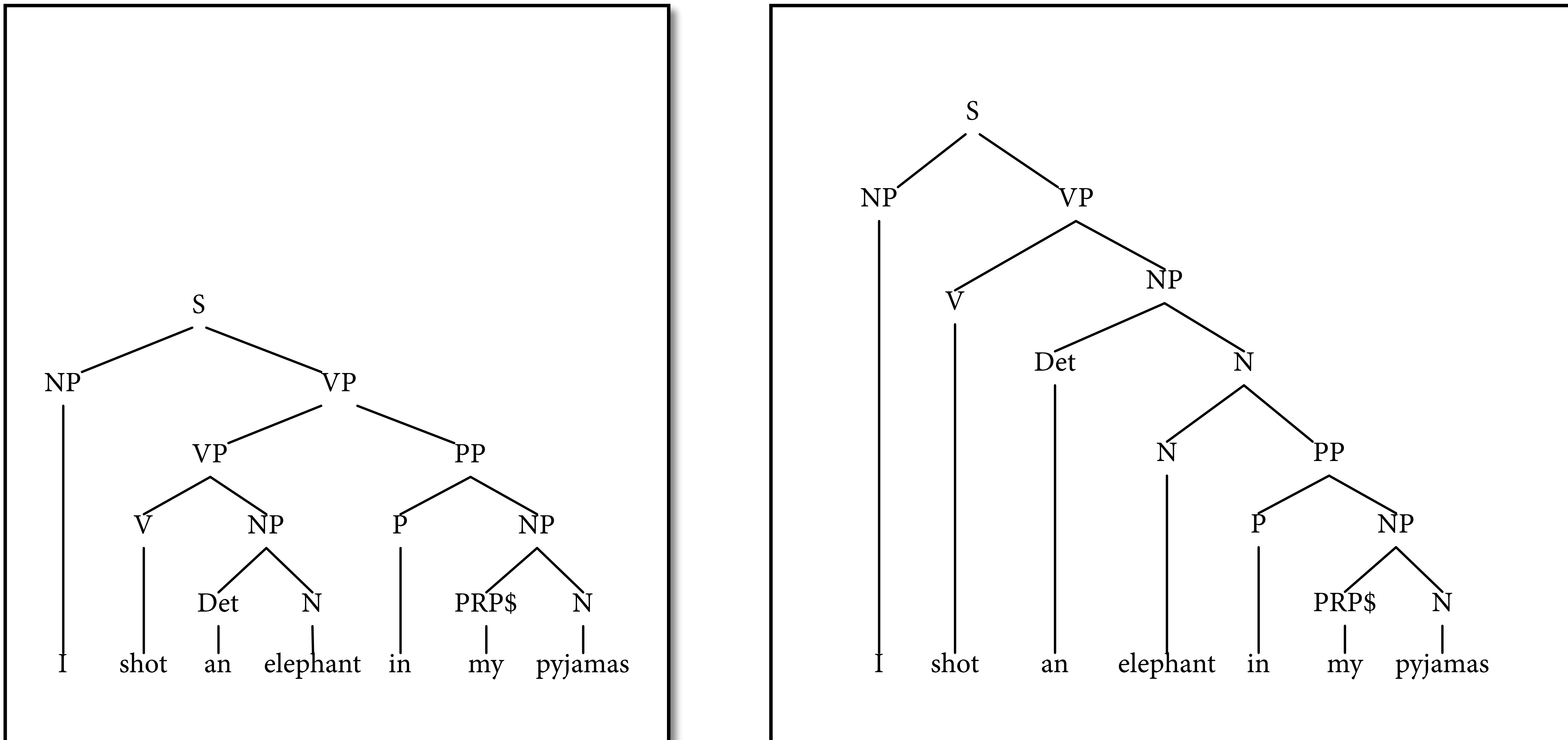
Sentences have structure

Record it conveniently in *phrase structure tree.*
Its inner nodes are *constituents* or *phrases.*



Ambiguity

Special challenge: sentences can have many possible structures.



This sentence is example of *attachment ambiguity.*

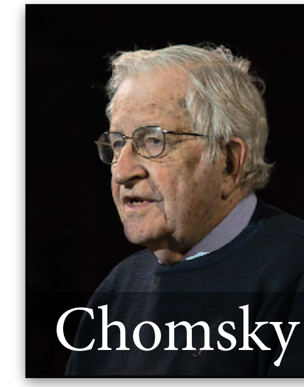
Grammars

- 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.
  - Note that syntactic parsing can also be done at high accuracy without grammars, e.g. using neural networks.

- Today we will 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 ∈ N is the *start symbol.*
  - P is a finite set of *production rules* of the form A → w,
    where A is nonterminal and w is a string from $(N \cup T)^*$.

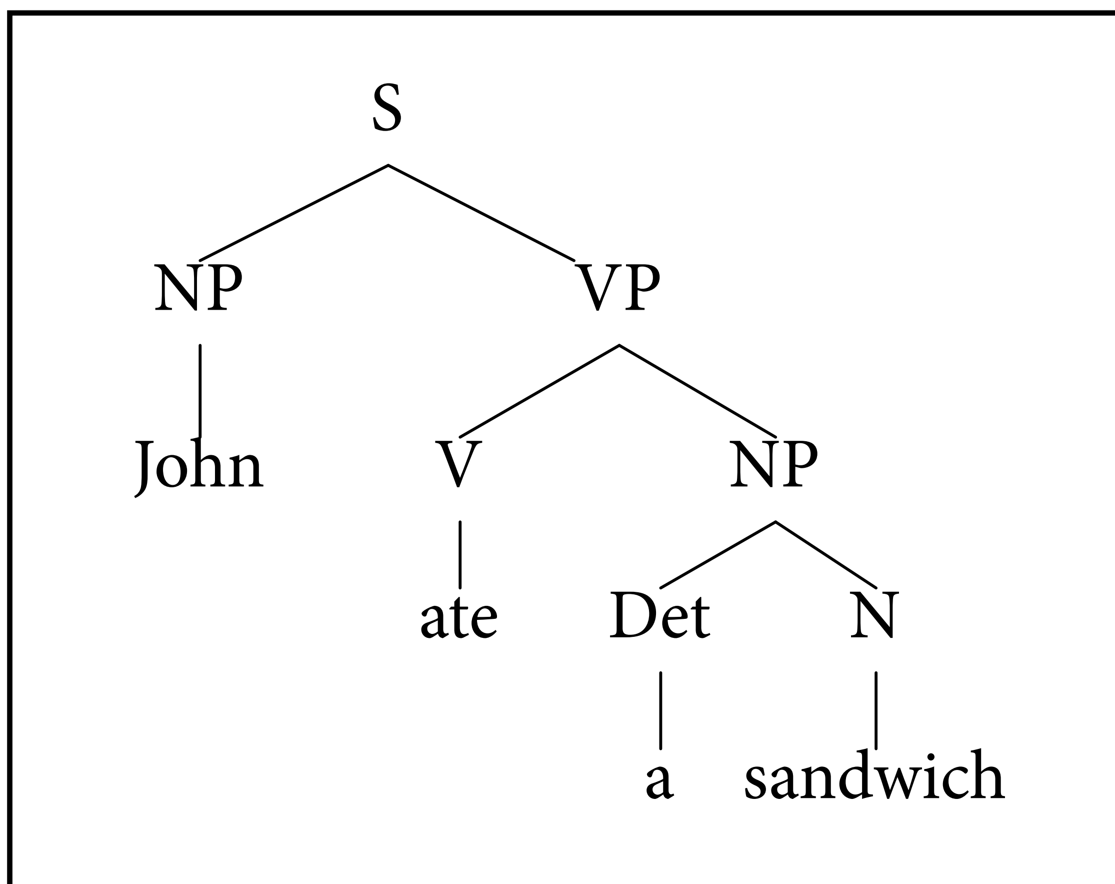
- Why “context-free”?
  - Left-hand side of production is a single nonterminal A.
  - Rule can’t look at context in which A appears.
  - *Context-sensitive* grammars can do that.

Example

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

Production rules:

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



Languages

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

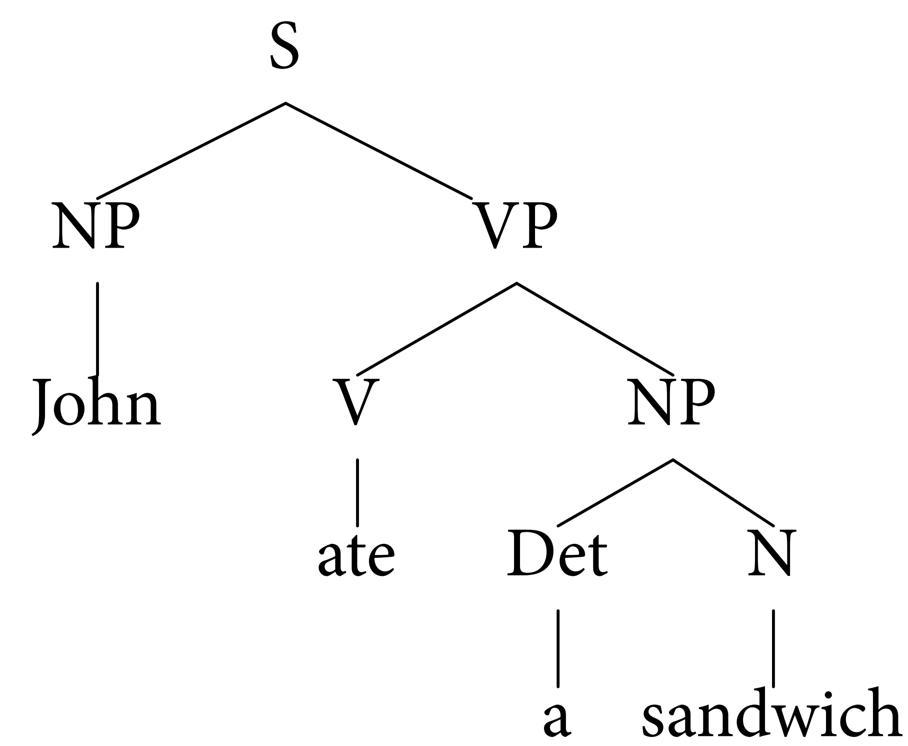
Derivations and parse trees

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

derivation

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

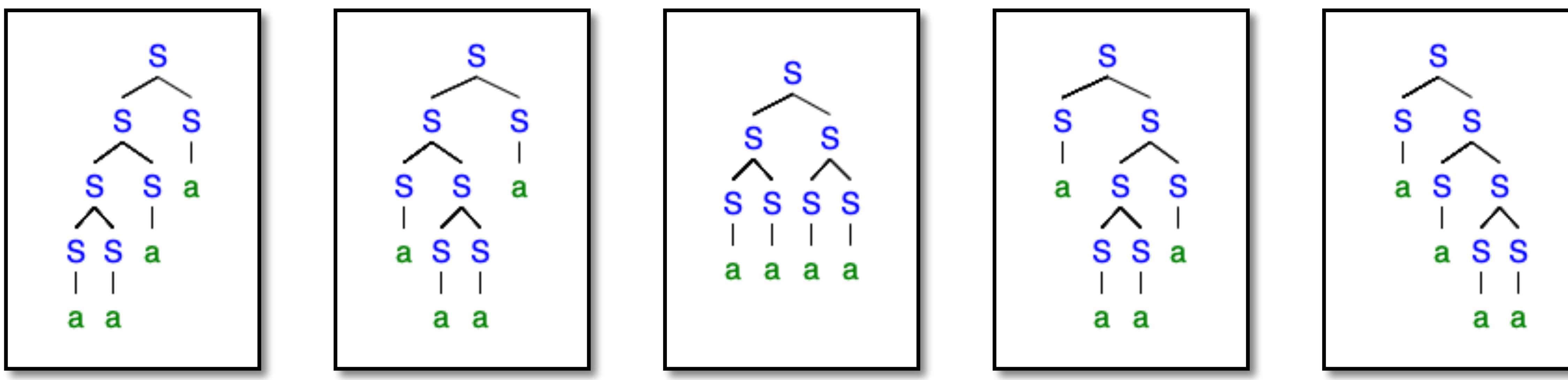
parse tree



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 ∈ 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

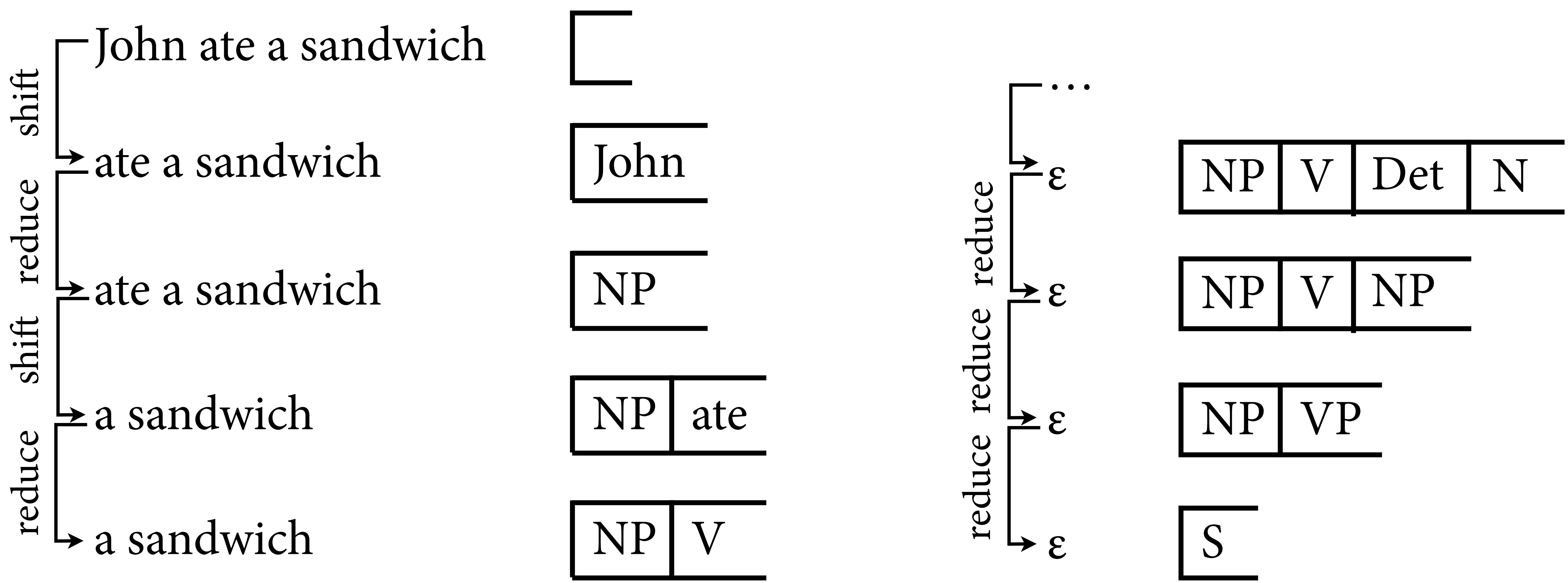
- How can we solve the word and parsing problem so systematically that we can implement it?
- One simple approach: shift-reduce algorithm (here: only for the word problem).
- Next time: Analyze efficiency of SR and replace it with faster algorithm: CKY.

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* 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 would need to try all sequences out.
  - Compilers for programming languages avoid this by careful language design: no ambiguity in grammar.

Interlude: Deduction and Logic

- Formulas in (propositional, predicate) logic make claims about a model.

- We can *infer* true statements by applying *deduction rules* to a set of true *axioms*, and thus try to *prove* where a hypothesis follows from the axioms.

- There are many different *proof systems* for doing this, e.g. modus ponens:

$$\frac{P \qquad P \rightarrow Q}{Q} \; (\mathrm{MP})$$

Example

Axioms

$R \to W$
"If it rains, the street is wet."

$W \to S$
"If the street is wet, then it is slippery."

R
"It is raining."

Hypothesis

S
"Is the street slippery?"

Proof system

$$\frac{P \qquad P \to Q}{Q} \text{ (MP)}$$

Proof

1	R	(Axiom)
2	$R \to W$	(Axiom)
3	W	(MP from 1, 2)
4	$W \to S$	(Axiom)
5	S	(MP from 3, 4)

Deductive Parsing

- Can we apply the same idea to parsing?
- Parsing algorithm derives claims about the string. Record such claims in *parse items*.
- At each step, apply a *parsing rule* to infer new parse items from earlier ones.
- If there is a way to derive a *goal item* from the *start item(s)* for a given input string, then claim that this string is in the language.

Parsing schema for shift-reduce

- Items are of the form (s,w') where w' is a suffix of the input string w, and s is the stack.
  - Claim of this item: Underlying cfg allows the derivation s w' $\Rightarrow^*$ w
  - Call item *true* if its claim is true.
- Start item (= axiom): (ε, w);
  goal item (= hypothesis): (S, ε)
- Parsing rules (= proof system):

$$\frac{(s,\ a \cdot w')}{(s \cdot a,\ w')}\ \text{(shift)} \qquad\qquad \frac{(s \cdot s',\ w') \quad A \rightarrow s' \text{ in } P}{(s \cdot A,\ w')}\ \text{(reduce)}$$

Shift-reduce schema: Example

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

1	(ε, John ate a sandwich)	Start	
2	(John, ate a sandwich)	shift, 1	
3	(NP, ate a sandwich)	reduce, 2, NP → John	
4	(NP ate, a sandwich)	shift, 3	
5	(NP V, a sandwich)	reduce, 3, V → ate	
6	(NP V a, sandwich)	shift, 5	
7	(NP V Det, sw.)	reduce, 6, Det → a	
8	(NP V Det sw., ε)	shift, 7	
9	(NP V Det N, ε)	reduce, 8, N → sw.	
10	(NP V NP, ε)	reduce, 9, NP → Det N	
11	(NP VP, ε)	reduce, 10, VP → V NP	
12	(S, ε)	reduce, 11, S → NP VP	

$$\frac{(s,\ a \cdot w')}{(s \cdot a,\ w')}\ \text{(shift)} \qquad \frac{(s \cdot s',\ w') \quad A \rightarrow s' \text{ in } P}{(s \cdot A,\ w')}\ \text{(reduce)}$$

Implementing schemas

- Can generally implement parser for given schema in the following way:
  - maintain an *agenda:* queue of items that we have discovered, but not yet attempted to combine with other items
  - maintain a *chart* of all seen items for the sentence

```
initialize chart and agenda with all start items

while agenda not empty:
  item = dequeue(agenda)
  for each new item it' that can be produced from item:
    if it' not already in chart:
      add it' to chart
      add it' to agenda

if chart contains a goal item, claim w ∈ L(G)
```

Agenda-based parsing

T = {John, sleeps}
N = {S, NP, VP, V}; start symbol: S

Production rules:

S → NP VP
NP → Det N
VP → sleeps
NP → John

New items it':

[illegible]

(John sleeps, ε)

Agenda:

(ε, John sleeps) (John, sleeps) (NP, sleeps) (John sleeps, ε) (NP sleeps, ε)

(John VP, ε) (NP VP, ε) (S, ε)

Chart:

(ε, John sleeps) (John, sleeps) (NP, sleeps) (John sleeps, ε) (NP sleeps, ε)

(John VP, ε) (NP VP, ε) (S, ε)

Correctness of shift-reduce

- Why should we believe that the SR parser always makes correct claims about the word problem?

- To convince ourselves, we need to prove:
  - *soundness:* SR recognizer only claims $w \in L(G)$ if this is true;
  - *completeness:* if $w \in L(G)$ is true, then SR recognizer claims it is.

- Proofs are easier if we assume that the cfg is in *Chomsky Normal Form:* every rule is of form $A \rightarrow a$ (exactly one terminal) or $A \rightarrow B\ C$ (exactly two nonterminals).

Soundness

- Show: If SR recognizer claims $w \in L(G)$, then it is true.

- Prove by induction over length k of SR derivation that all items that SR derives from start item are true.

(1) $\quad (\varepsilon, w) \rightarrow^{k-1} (s, a w') \xrightarrow{\text{shift}} (s a, w')$

$$s (a w') \Rightarrow^* w \quad \Rightarrow \quad (s a) w' \Rightarrow^* w$$

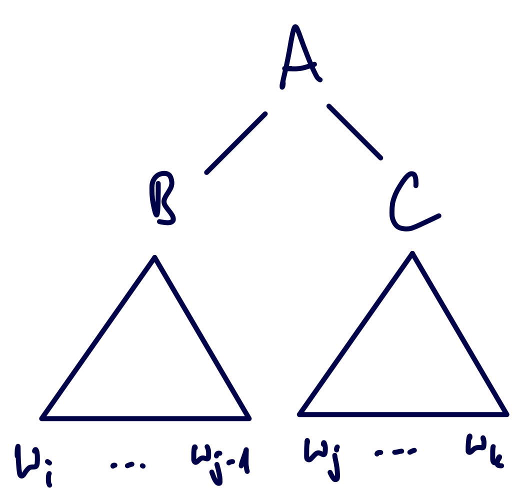
(2) $\quad (\varepsilon, w) \rightarrow^{k-1} (s s', w') \xrightarrow{\text{reduce}} (s A, w')$

$$s s' w' \Rightarrow^* w \quad \Rightarrow \quad s A w \Rightarrow s s' w' \Rightarrow^* w$$

Completeness

- Show: If w ∈ L(G), then SR recognizer claims it is true.
- Prove by induction over length of CFG derivation that if $A \Rightarrow^* w_i \ldots w_k$, then $(\varepsilon, w_i \ldots w_k) \xrightarrow[SR]{}^* (A, \varepsilon)$.

$A \Rightarrow^* w_i \ldots w_k$
looks like this if $k > i$:



Thus, by induction hypothesis:

$$
\begin{aligned}
&(\varepsilon, w_i \ldots w_k) \\
&\xrightarrow[SR]{}^* (B, w_j \ldots w_k) \\
&\xrightarrow[SR]{}^* (B\,C, \varepsilon) \\
&\xrightarrow[reduce]{} (A, \varepsilon)
\end{aligned}
$$

Conclusion

- Context-free grammars: most popular grammar formalism in NLP.
- Parsing algorithms.
  - today, shift-reduce
  - next time, CKY
- Outlook:
  - combine CFG parsing with statistics → PCFGs
  - more expressive grammar formalisms
  - neural and neurosymbolic parsing algorithms