Lexicalized grammar formalisms

Computational Linguistics

Alexander Koller

02 January 2024

Outline

1. Tree Substitution Grammars
2. Tree Adjoining Grammars (TAG)
3. Combinatory Categorial Grammars (CCG)
4. Supertagging

Grammar-based parsing

- Context-free grammars have many strengths.
  - simple parsing algorithms with decent complexity
  - simple but effective probability model (PCFGs)
- They also have weaknesses.
  - Some phenomena in NL syntax not context-free; can't be correctly modeled with (P)CFGs.
  - Grammars cannot be *lexicalized*; hard to predict syntactic structure in which a word is used.
- Let's look at these in more detail.

Some NLs not context-free

… because they allow *cross-serial dependencies*.

Jan säit das mer es huus haend wele aastriiche.

Jan säit das mer d'chind es huus haend wele laa aastriiche.

Jan säit das mer em Hans es huus haend wele hälfe aastriiche.

Jan säit das mer d'chind em Hans es huus haend wele laa hälfe aastriiche.

* Jan säit das mer em Hans d'chind es huus haend wele laa hälfe aastriiche.

* Jan säit das mer em Hans em Sepp es huus haend wele laa hälfe aastriiche.

* Jan säit das mer em Hans es huus haend wele laa hälfe aastriiche.

$\Rightarrow$ w a^m b^n x c^m d^n y $\Rightarrow$ not context-free

(Shieber 1985 on Swiss German; everybody should read this paper)

Lexicalization

- We call a grammar *lexicalized* if every piece of grammatical information is tied to a word.
  - For every word, there is a finite set of lexicon entries which say how it can combine with others.

- Advantages:
  - convenient in manual grammar development
  - can make parsing really fast (supertagging)
  - pick up ideas from dependency parsing
  - grammar captures long-distance relationships between words in a linguistically clean way

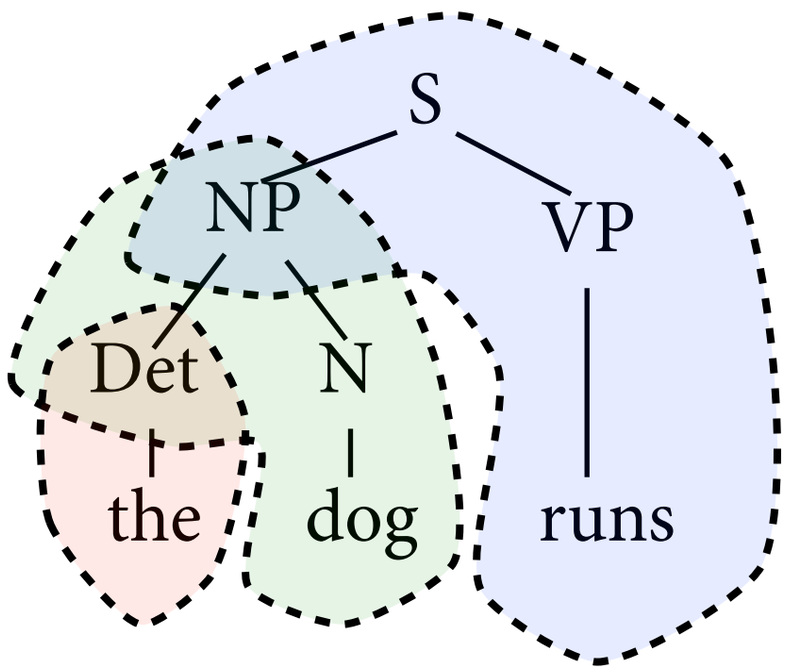
CFGs are not lexicalized

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

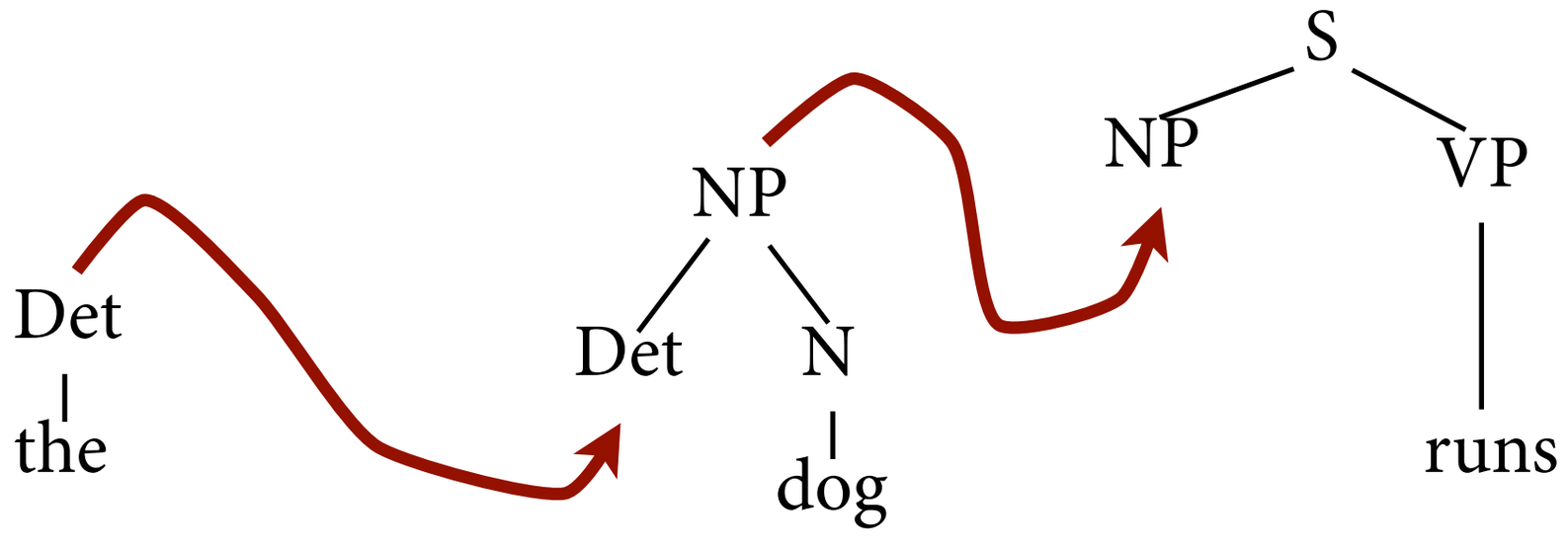
Production rules:

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

An attempt



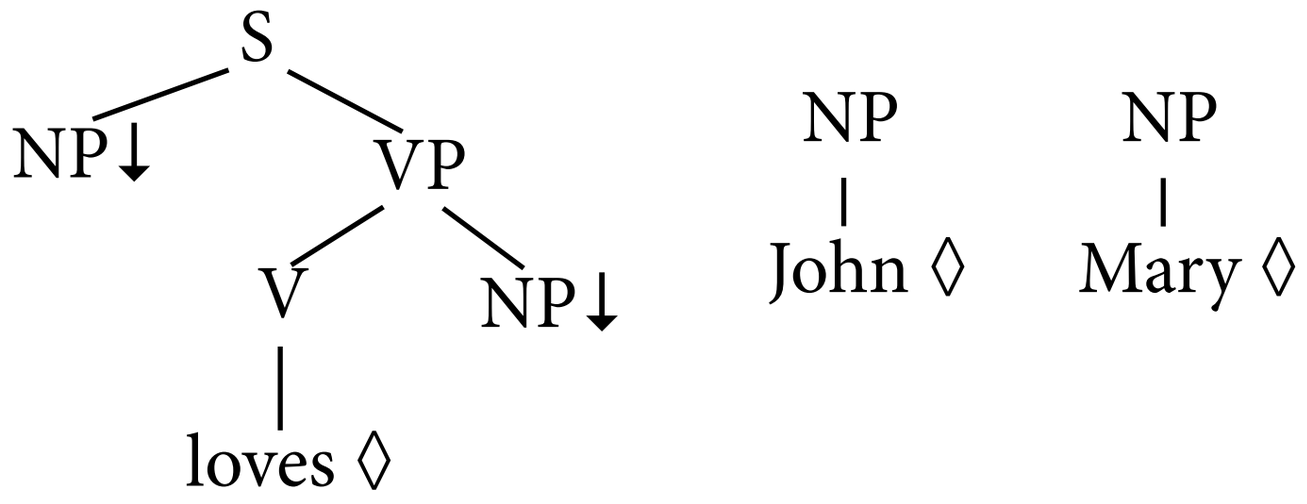
Can combine tree parts by plugging them into each other:



Tree substitution grammars

- *Tree substitution grammars (TSGs):*
  finite set of *elementary trees* for the words.

- Nodes of elementary trees:
  - *internal nodes*, labeled with nonterminal symbols
  - *lexical anchors,* leaves labeled with words or POS tags
    (sometimes marked with diamond: A◊)
  - *substitution nodes*, leaves marked with nonterminals
    (usually marked with downarrow: A↓)



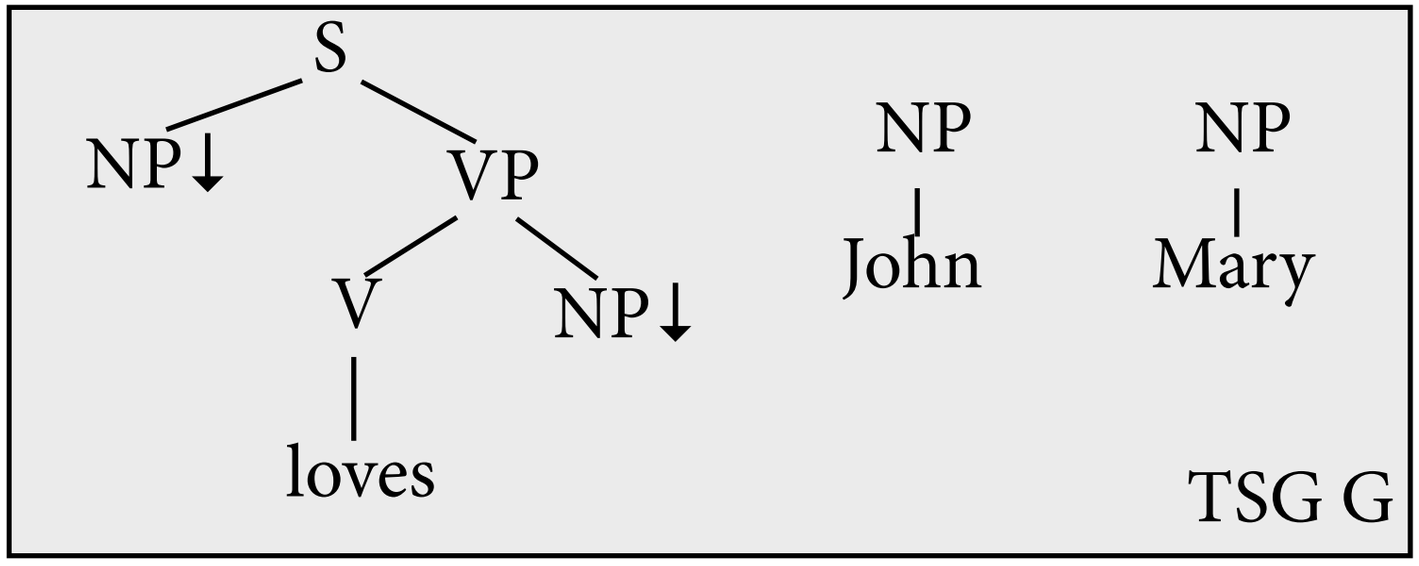
TSGs: Derivations

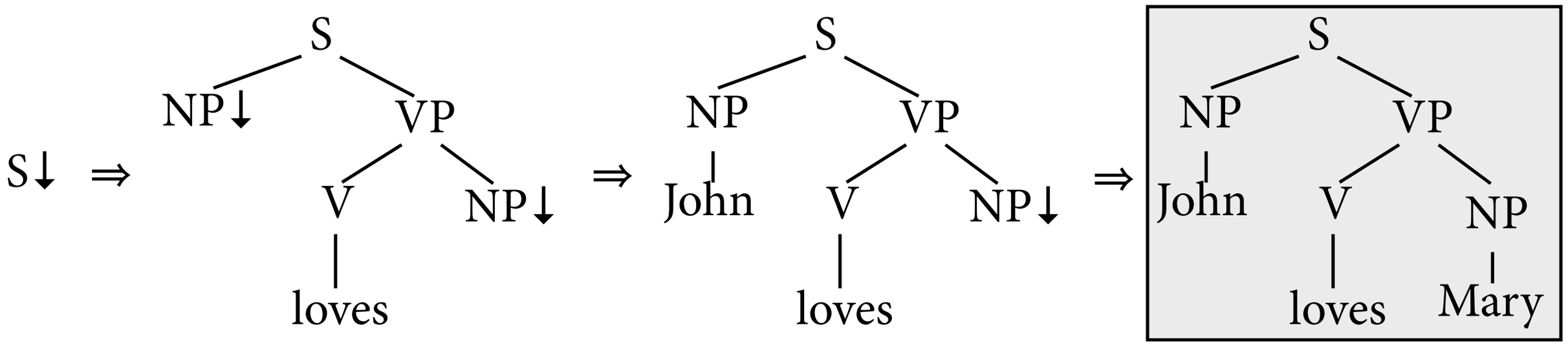
- Derivation step, $t \Rightarrow t'$:
  - t a tree that contains a substitution node u with label A
  - e an elementary tree with root label A
  - obtain t' from t by replacing u with e

- Derived tree t of grammar G: $S\downarrow \Rightarrow^* t$
  and contains no more substitution nodes.

- TSG describes:
  - language of derived trees
  - language of strings (= yields of derived trees)

Example derivation





no more substitution nodes,
therefore derived tree of grammar

Lexicalized TSG

- Call elementary tree lexicalized if it contains a lexical anchor.

- Call TSG lexicalized if all elementary trees are lexicalized.

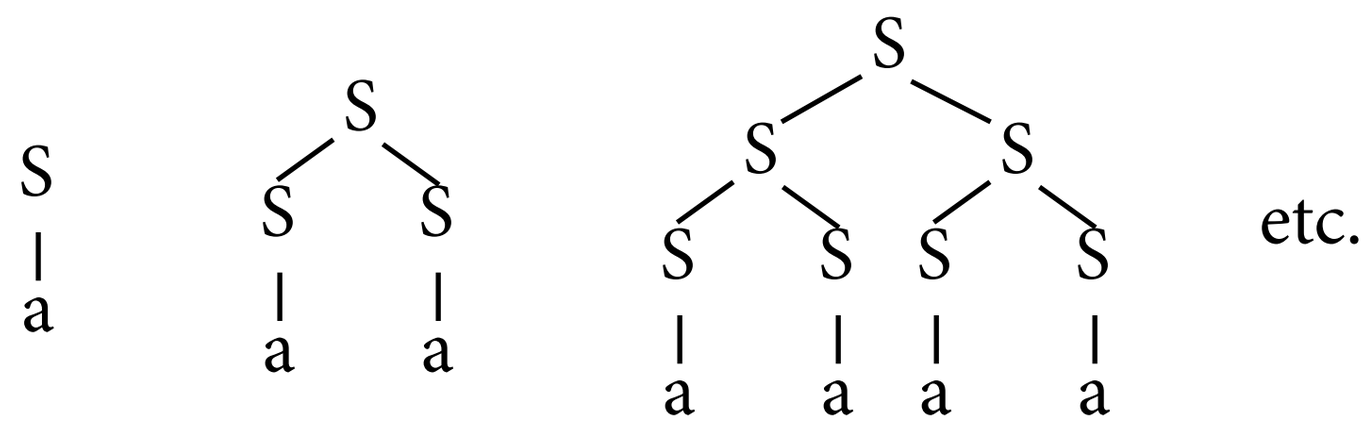
- Can we strongly lexicalize all CFGs into TSGs?
  - that is: is it true that for every CFG, there is a lexicalized TSG such that derived trees of TSG = parse trees of CFG?

No!

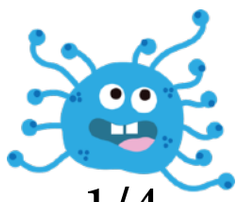
- Counterexample (Schabes):

$$\mathrm{S} \rightarrow \mathrm{S}\ \mathrm{S}$$
$$\mathrm{S} \rightarrow a$$

- Path to highest leaf can be arbitrarily long; but is bounded in any given lexicalized TSG.

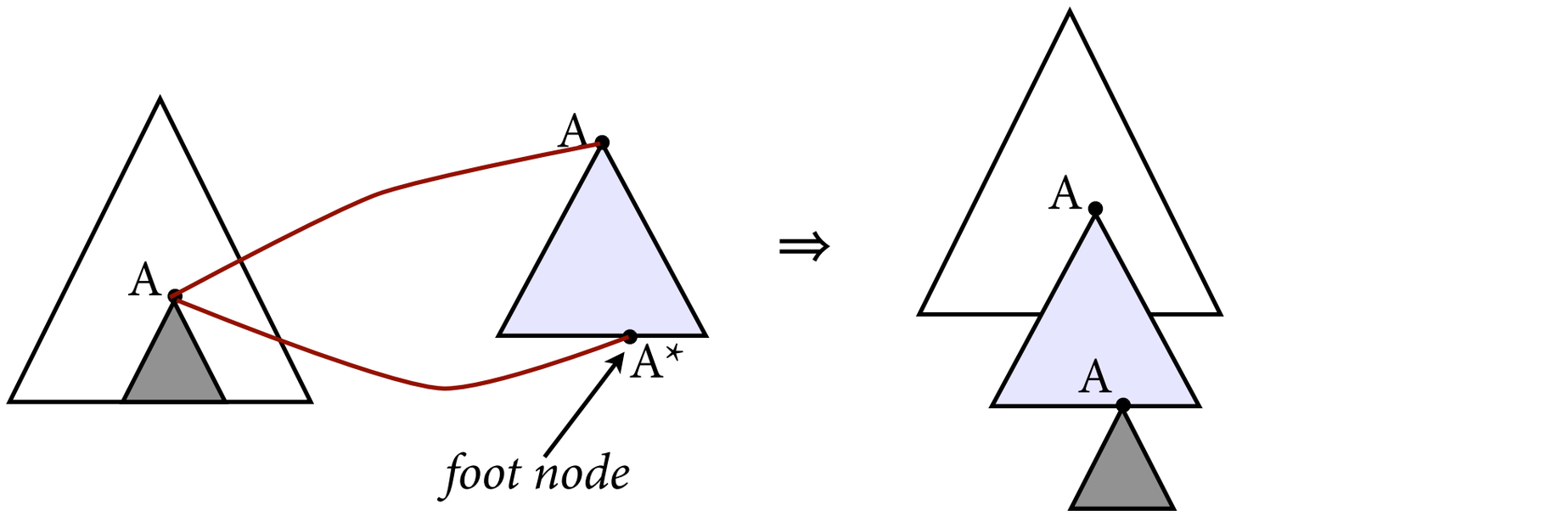


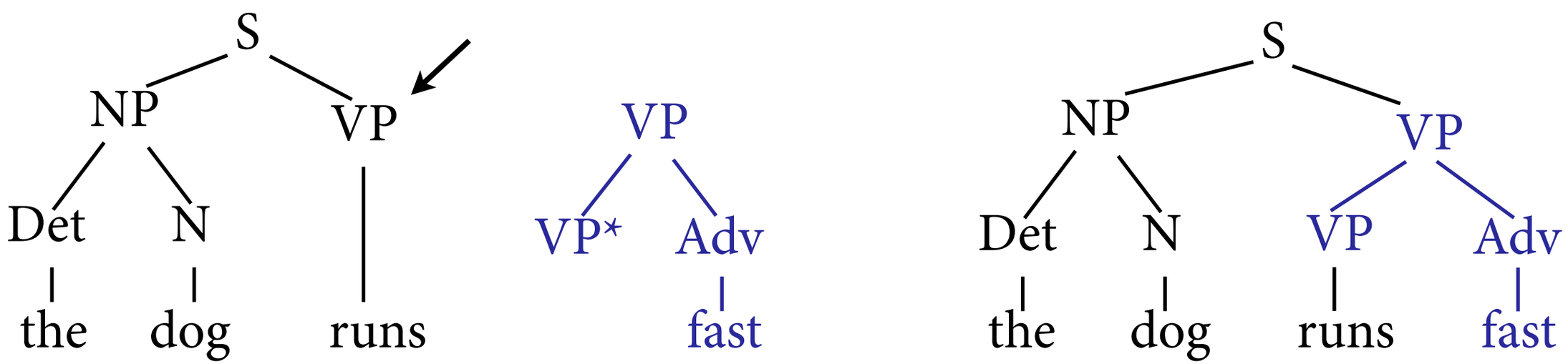
etc.



Adjunction

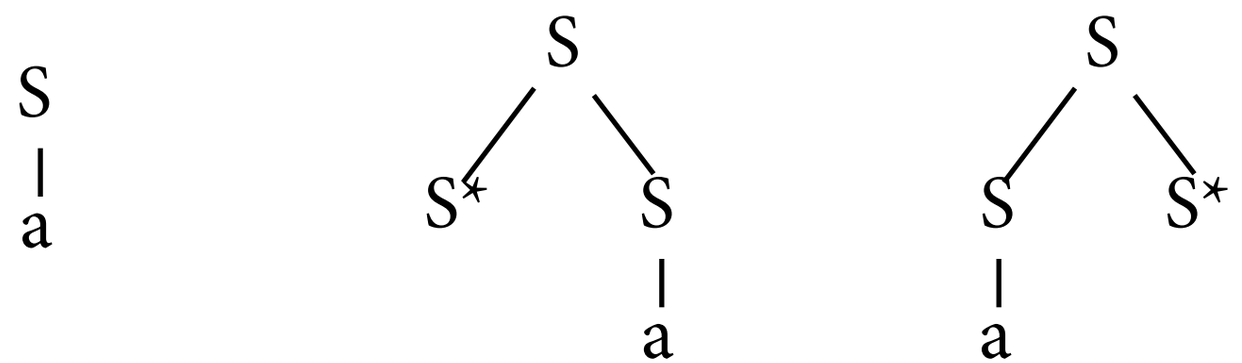
- To lexicalize example CFG, we can use a second tree-combining operation: *adjunction*.



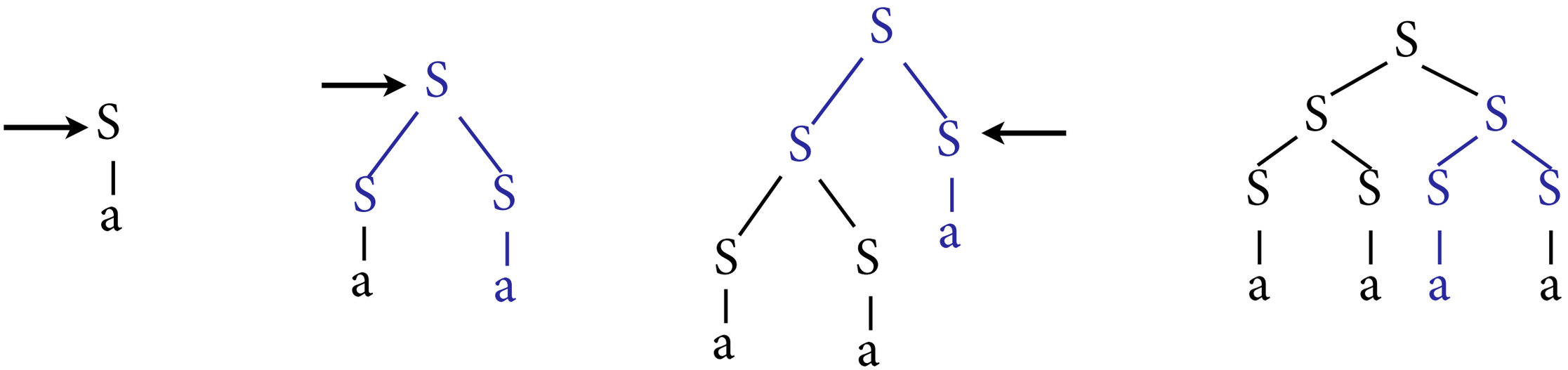


Lexicalization with adjunction

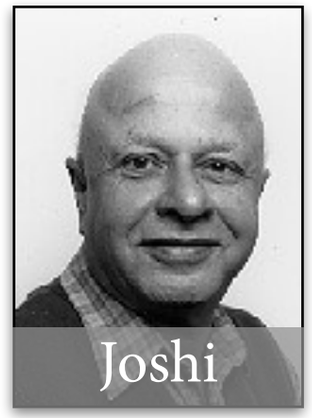
- Using these lexicalized elementary trees:



- … we can build all parse trees of the example CFG:



Tree-adjoining grammars


Joshi

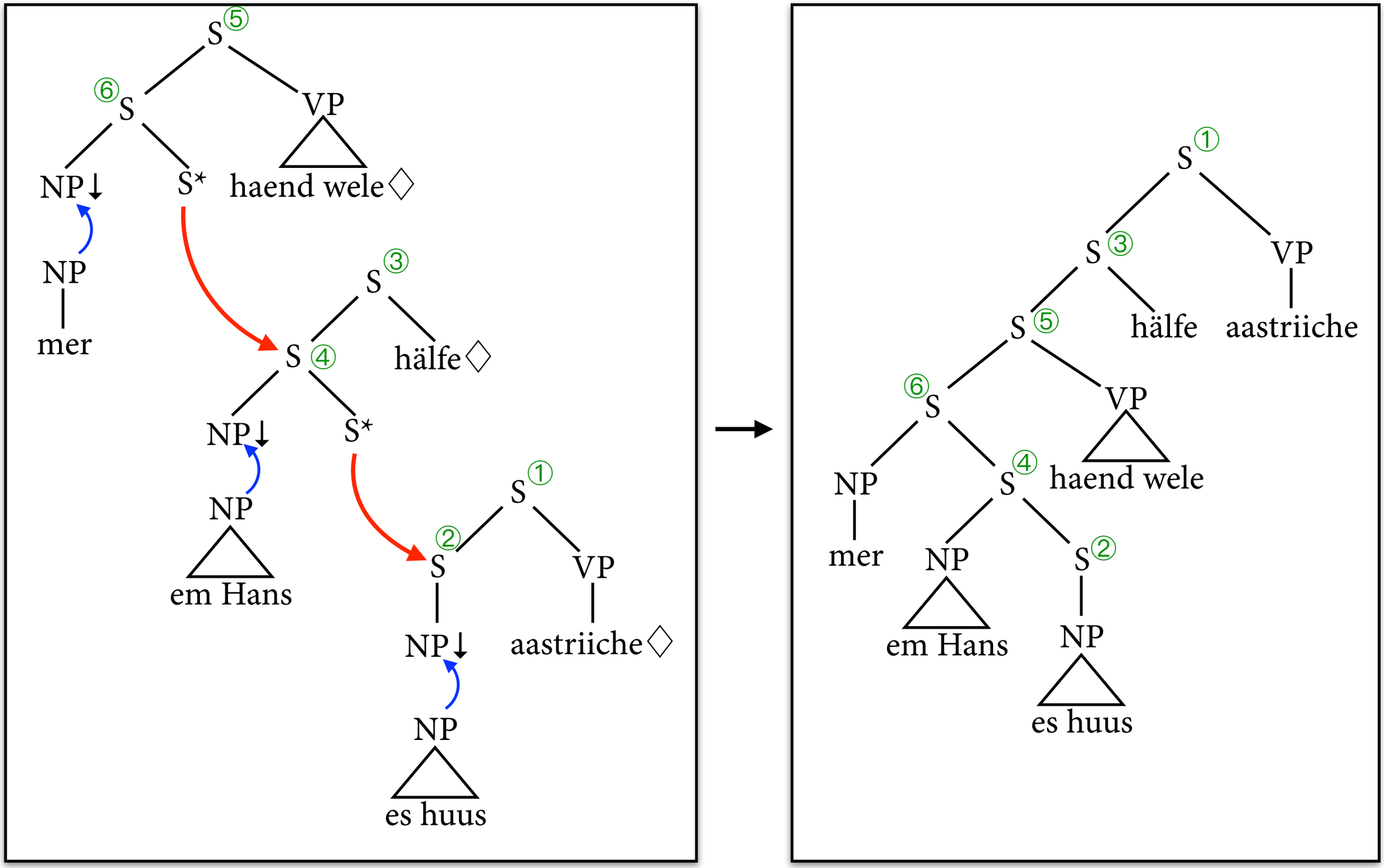
- A (lexicalized) TAG grammar consists of a finite set of lexicalized elementary trees.
  - *initial trees:* have no foot node
  - *auxiliary trees:* have foot nodes

- Combine these using substitution and adjunction.

- Can prove that for every CFG, there is a strongly equivalent, lexicalized TAG grammar.

- In addition to all context-free grammars, can also describe languages that are not context-free.

Swiss German in TAG



Parsing

- Can define a CKY-style parser for TAG. Items:
  - [A, i, k, …] for substring from i to k
    (yield of initial trees)
  - <A, i, j, k, l, …> for *pair* of substrings from i-j and k-l
    (yield of auxiliary trees)

- Most expensive rule wraps one pair around another:

$$\frac{\langle A, i_1, i_2, i_5, i_6, \beta_1, \epsilon \rangle \quad \langle A, i_2, i_3, i_4, i_5, \beta_2, \pi \rangle}{\langle A, i_1, i_3, i_4, i_6, \beta_2, \pi \rangle}$$

- Thus, parsing complexity $O(n^6)$.

For more details, see e.g. Kallmeyer (2010), Parsing Beyond Context-Free Grammars

Categorial Grammars

- CFG and TAG based on *phrase structure grammar*:
  - combine small constituents into larger constituents
  - finite set of nonterminals {V, NP, …} with no inherent relationship; grammar says how they can be combined

- One alternative: *categorial grammar.*
  - NL expressions have *categories*, e.g. S\NP
  - category says what type of substrings it would like to be combined with (NP) and what type of substring this will produce (S) ⇒ functor-argument structure made explicit
  - … and on what side we're looking for the NP
    (slash "/" = to the right; backslash "\" = to the left)

Example

$$\dfrac{\dfrac{\text{John}}{\text{NP}}\quad\dfrac{\dfrac{\text{eats}}{(\text{S}\backslash\text{NP})/\text{NP}}\quad\dfrac{\dfrac{\text{a}}{\text{NP}/\text{N}}\quad\dfrac{\text{sandwich}}{\text{N}}}{\text{NP}}}{\text{S}\backslash\text{NP}}}{\text{S}}$$

CCG

- Combinatory Categorial Grammar (CCG): one grammar formalism for CG which is popular in computational linguistics.

- Grammar specifies:
  - finite set of categories for each word in the lexicon
  - rules for combining categories (application, composition, type-raising)

Application

- Application rules combine a functor category with the next argument that it is looking for.

$$\frac{X/Y \quad Y}{X} >$$

forward application

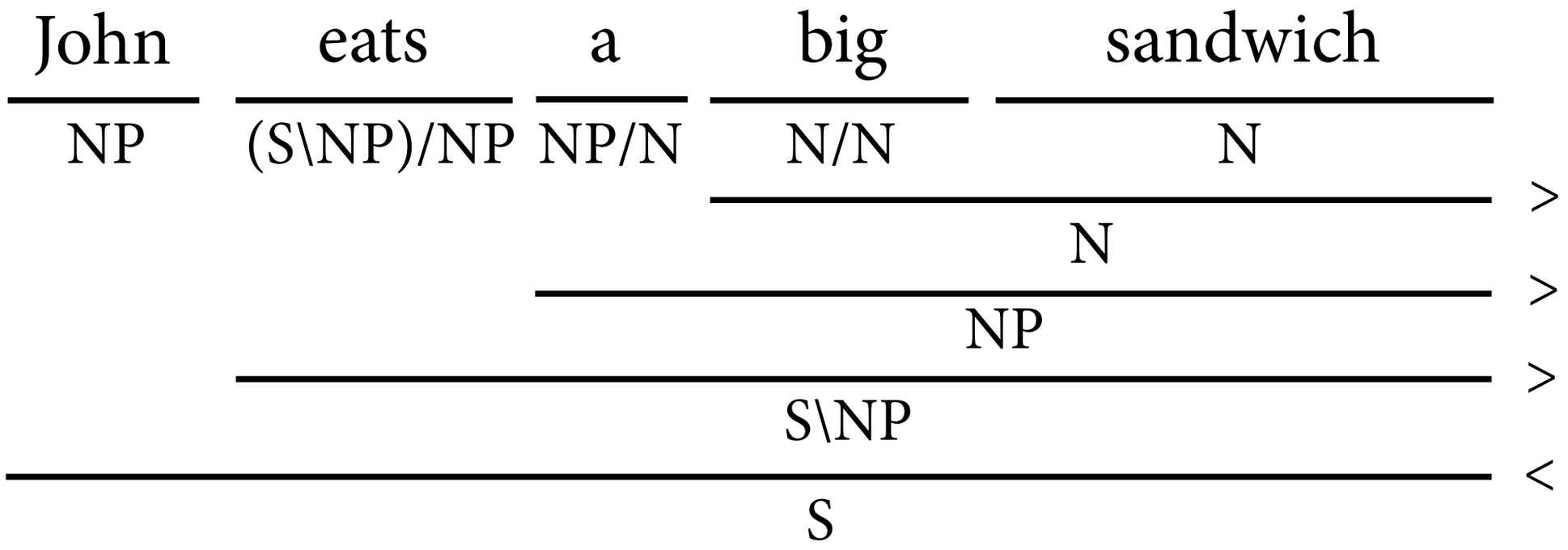
$$\frac{Y \quad X\backslash Y}{X} <$$

backward application

Example

Lexicon:

John: NP	a: NP/N
eats: (S\NP)/NP	sandwich: N
big: N/N	



Composition

- Using the *composition* rules, can combine two categories and pass "leftover" arguments to the category of the bigger string.

Harmonic composition:

$$\frac{X/Y \quad Y/Z}{X/Z} >B \qquad \frac{Y\backslash Z \quad X\backslash Y}{X\backslash Z} <B$$

Crossed composition:

$$\frac{X/Y \quad Y\backslash Z}{X\backslash Z} >Bx \qquad \frac{Y/Z \quad X\backslash Y}{X/Z} <Bx$$

Swiss German

- Crossed composition allows us to model cross-serial dependencies.*

$$\dfrac{\text{das}\quad \underbrace{\text{mer}}_{NP_{nom}}\quad \underbrace{\text{em Hans}}_{NP_{dat}}\quad \underbrace{\text{es huus}}_{NP_{acc}}\quad \underbrace{\text{hälfed}}_{S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}/VP}\quad \underbrace{\text{aastriiche}}_{VP\backslash NP_{acc}}}{\dfrac{\dfrac{\dfrac{S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}\backslash NP_{acc}}{S_{+SUB}\backslash NP_{nom}\backslash NP_{dat}}<}{S_{+SUB}\backslash NP_{nom}}<}{S_{+SUB}}<}>Bx$$

(“VP” = abbreviation for $S\backslash NP_{nom}$)

*) with certain limitations, see Kuhlmann, Koller, Satta 2015

Some formal results

- Can show that certain versions of TAG and CCG are weakly equivalent: generate same language class.
  - proof by Vijay-Shanker and Weir, early 1990s
  - requires CCG grammars that are not entirely lexicalized (Kuhlmann et al. 2015)

- Therefore, word problem of CCG is $O(n^6)$.

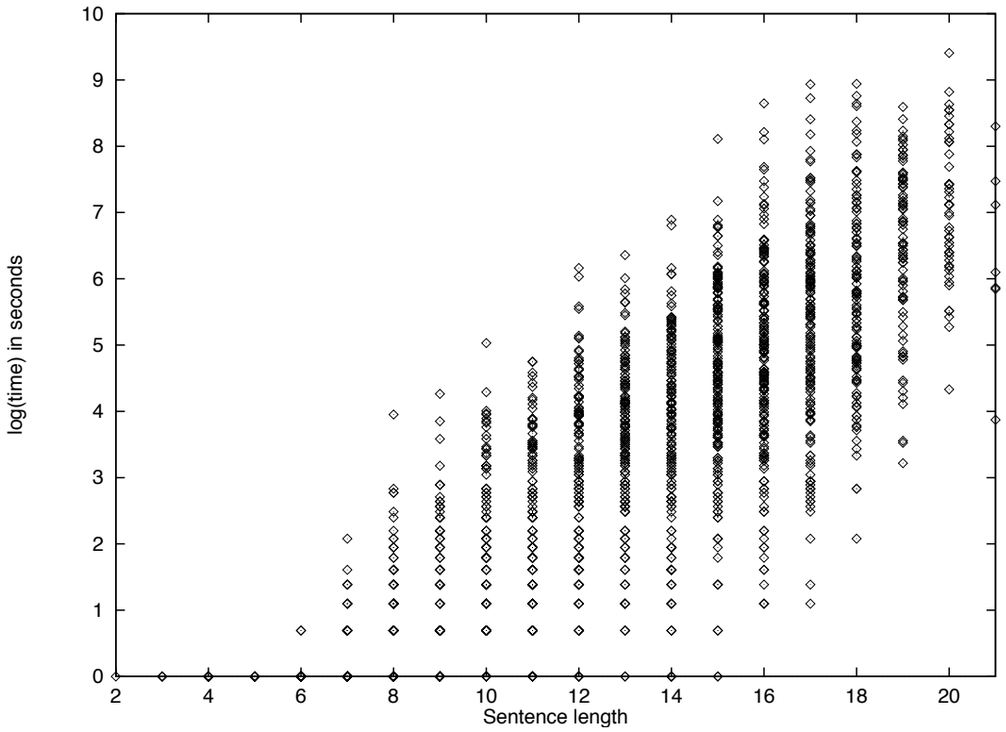
- Polynomial CCG parser is a hassle to implement; most implementations (e.g. OpenCCG, C&C) are worst-case exponential.

Supertagging

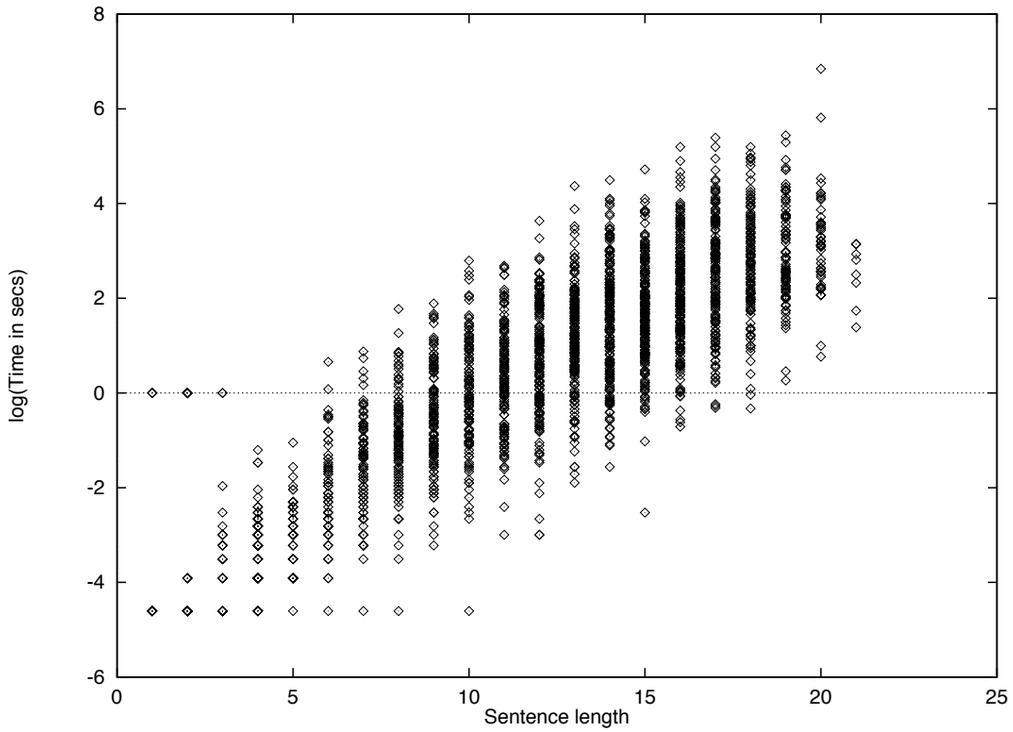
- Practical parsing time is determined by degree of *lexical ambiguity:* how many lexicon entries per word?
  - This is worse for TAG than for CCG because CCG's combination operations more flexible than TAG's.
  - Not unusual to have hundreds of lexicon entries per word in large-scale TAG grammar.

- Supertagging: use statistical methods to narrow down lexicon entries before parsing starts.
  - Use methods for other tagging tasks (e.g. POS tagging): e.g. HMMs, CRFs, neural networks.

Supertagging



TAG parsing without supertagging



k-best supertagging with k=60

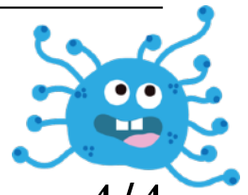
(Sarkar 07)

State of the art

- CCG parser of Lewis et al. (2016):
  - very accurate supertagger based on LSTMs
  - drastically simplified probability model (supertag-factored)
  - very fast parsing through A* search and parallelization

Model	QUESTIONS			BIOINFER		
	P	R	F1	P	R	F1
C&C	-	-	86.6	77.8	71.4	74.5
EASYCCG	78.1	78.2	78.1	76.8	77.6	77.2
C&C + RNN	-	-	-	80.1	75.5	77.7
LSTM	87.6	87.4	87.5	80.1	80.9	80.5
LSTM + Dependencies	**88.2**	**87.9**	**88.0**	77.8	80.1	79.4
LSTM + Tri-training	-	-	-	**81.8**	**82.6**	**82.2**

Parser	**Sentences per second**
SpaCy*[4]	778
Berkeley GPU* (Hall et al., 2014)	687
Chen and Manning (2014)*	391
C&C	66
EASYCCG	606
LSTM	214
LSTM + Dependencies	58
LSTM GPU	2670



Conclusion

- Rich literature on grammar formalism that are more expressive than CFGs.
  - expressive capacity needed for some linguistic phenomena
  - more convenient for manual grammar development
  - lexicalization
  - see Syntactic Theory lecture if you want more details

- Parsing for these formalisms:
  - higher asymptotic complexity than for CFGs
  - if done right, supertagging for lexicalized grammars can yield extremely fast parsers