Syntax and formal grammars

Benoît Sagot

Inria (ALMAnaCH)

MVA — Speech and Language Processing — Class #5 — 17th February, 2020

What is syntax?

- **Syntax** is the branch of linguistics that studies the **structural properties of sentences**
  - It does not study the structural properties of **wordforms**, which are by definition the **atoms of syntax**
  - It does not directly study the consequence of the semantic properties of word(form)s, but it studies how such properties are dealt with in syntax
  - Syntax is heavily **language-dependent**
  - Syntax is heavily "**lexicalised**"

Syntax and NLP

- In NLP, **syntax** is at the crossroads between:
  - **syntax** *per se*, i.e. the branch of linguistic that studies syntax:
    what structures do we need to represent?
    - introspection
    - corpus studies
    - psycholinguistics / neurolinguistics
  - **formal grammars**, a branch of theoretical computer science:
    what formal devices do we need to represent such structures?
  - **language resources:**
    - syntactically annotated corpora, a.k.a. **treebanks**:
      can we build collections of sentences annotated with syntactic structures?
    - **syntactic lexicons**:
      can we describe the syntactic properties of wordforms/lexemes?

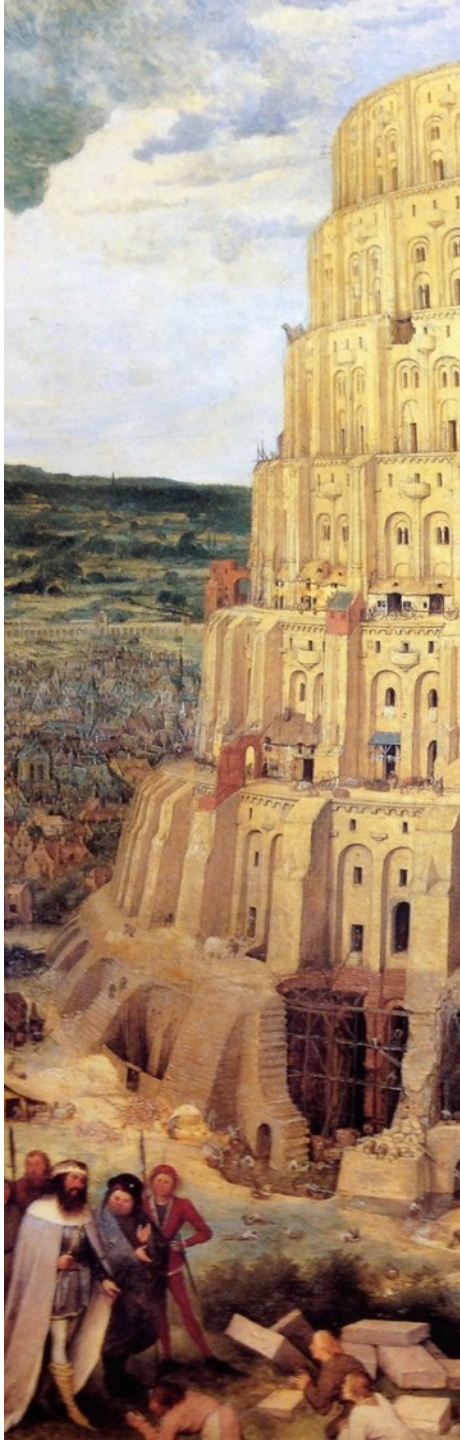
Parsing

- To build the syntactic structure of a sentence = to **parse** the sentence
- If syntactic structures are represented in the form of a tree, the syntactic structure of a sentence = its **parse tree**
- Producing the parse of a sentence = **parsing**
- “Parsing” can be used in more complex terms (e.g. semantic parsing)
  - By default, parsing = syntactic parsing
- A computer program for parsing is called a **parser**

- **Parsing** is a key step towards the computational exploitation of a sentence or a text. Most NLP tasks rely on or benefit from parsing, although it is not always easy to do so.

- **Parsing will be the topic of the next class**

Syntactic structures



Structures in trees

- **Grammatical** sentences:
  - *the boy likes a girl*
  - *the small girl likes the big girl*
  - *a very small nice boy sees a very nice boy*
- **Ungrammatical** sentences:
  - **the boy the girl*
  - **small boy likes nice girl*
- Can we find a way of distinguishing the two kinds of sequences?
- Can we identify similarities among grammatical subsequences?

A first version of constituent structure

- Underlying idea: sequences of words belonging together form **constituents**
- First attempt
  - Grammatical sentences:
    - *(the) boy (likes a girl)*
    - *(the small) girl (likes the big girl)*
    - *(a very small nice) boy (sees a very nice boy)*
  - Ungrammatical sentences:
    - **(the) boy (the girl)*
    - **(small) boy (likes nice girl)*

A second version of constituent structure

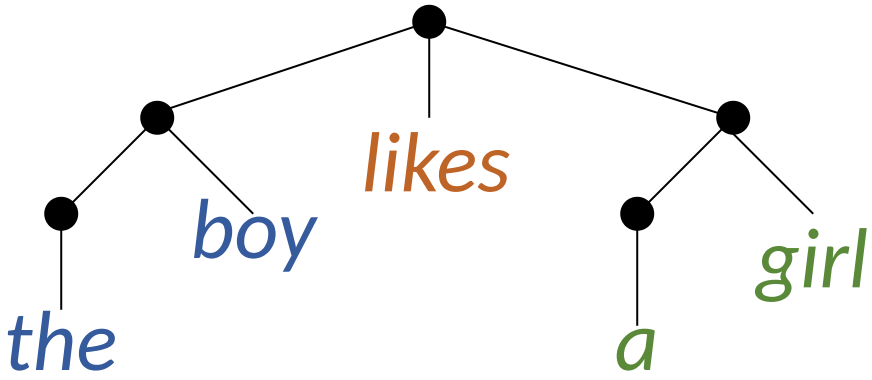
- Underlying idea: sequences of words belonging together form **constituents**
- Second attempt
  - Grammatical sentences:
    - *(the boy) likes (a girl)*
    - *(the small girl) likes (the big girl)*
    - *(a very small nice boy) sees (a very nice boy)*
  - Ungrammatical sentences:
    - **(the boy) (the girl)*
    - **(small boy) likes (nice girl)*
- Intuition: this version is better because it uses fewer types of constituents (blue and green are of same type)

More structures

- Underlying idea: sequences of words belonging together form **constituents**, in a **hierarchical** way
  - Grammatical sentences:
    - *((the) boy) likes ((a) girl)*
    - *((the) (small) girl) likes ((the) (big) girl)*
    - *((a) ((very) small) (nice) boy) sees ((a) ((very) nice) boy)*
  - Ungrammatical sentences:
    - **((the) boy) ((the) girl)*
    - **((small) boy) likes ((nice) girl)*

Constituency trees: a first attempt

- *((the) boy) likes ((a) girl)*



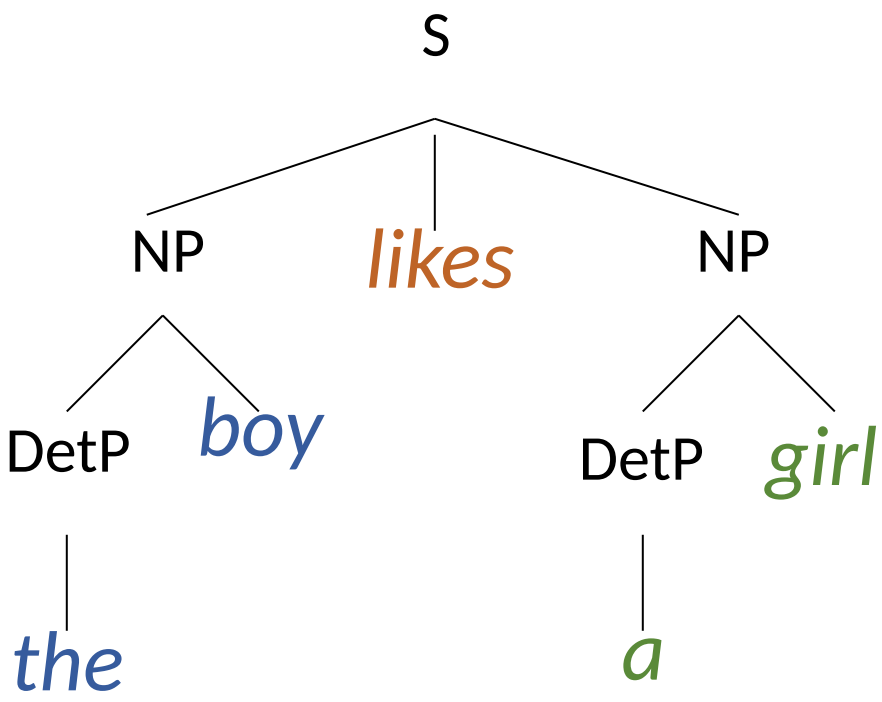
How do we label the nodes?

- *(((the) boy) likes ((a) girl))*

- Build constituents so each one has exactly one non-bracketed word, called its **head**
- Cluster constituents and word(form)s in a consistent way: words within the same cluster should head similar constituents
  - Wordform cluster = its **syntactic category**, or **part-of-speech (PoS)**
    - e.g. N(oun), V(erb), ADV(erb), ADJ(ective), DET(erminer), PREP(osition)…
  - Constituent type = XP(hrase), where X = PoS of its head
    - e.g. NP = noun phrase, AdjP = adjectival phrase…

Constituency trees: a first attempt

- *((the/DET) boy/N) likes/V ((a/DET) girl/N)*

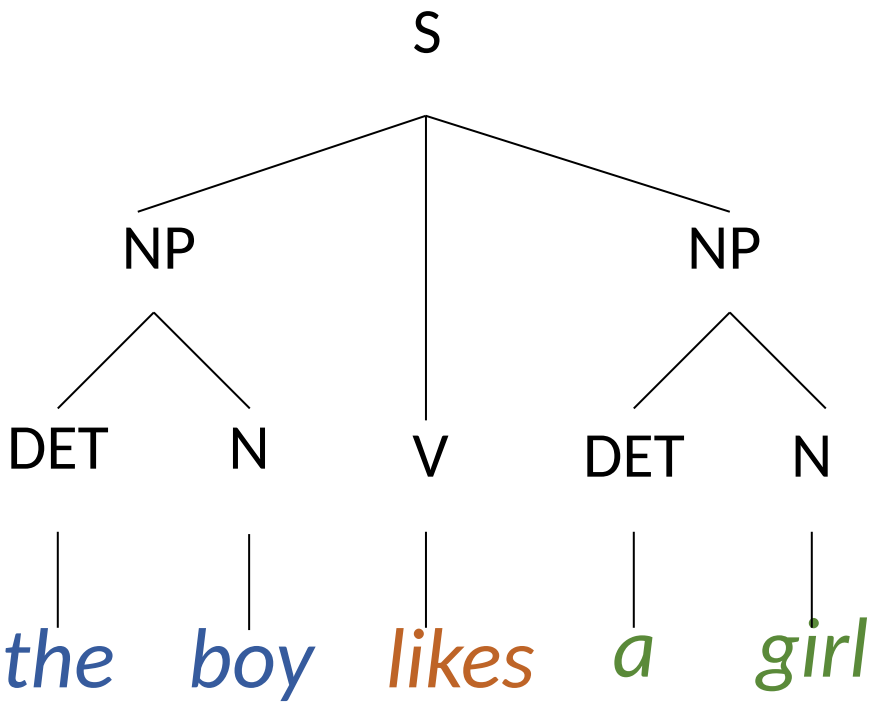


Constituency trees

- The “1 non-bracketed wordform per constituent” constraint is not applicable in practice
- The perfect symmetry between PoS and constituent types is a linguistic decision, not shared by everyone
  - We do not want to limit ourselves to one particular type of linguistic theory regarding syntax.
- It is useful and therefore common practice to insert **PoS as immediate ancestors of leaf nodes** (i.e. wordforms)
- We temporarily lose the notion of constituent head
  - It will come back in a few slides

Constituency trees: a second attempt

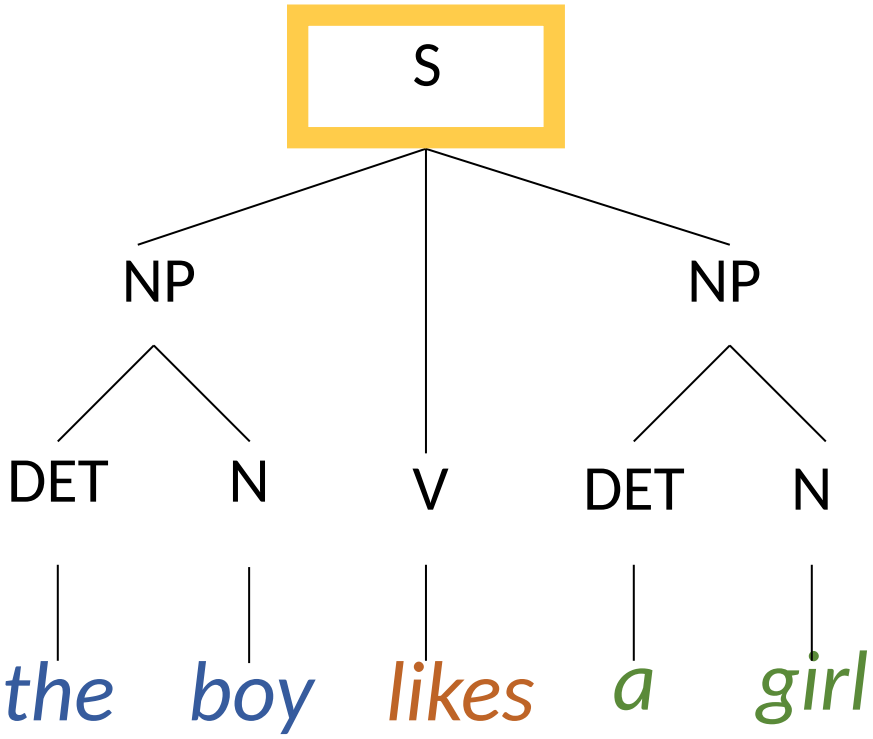
- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*



Constituency trees: a second attempt

- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*

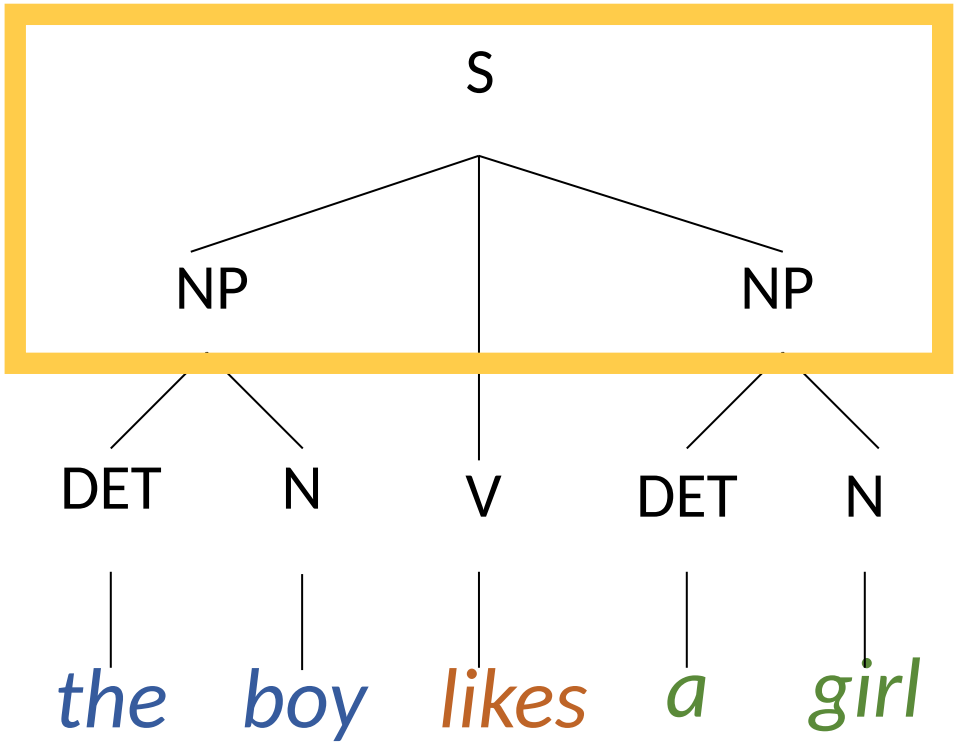
Root of the tree



Constituency trees: a second attempt

- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*

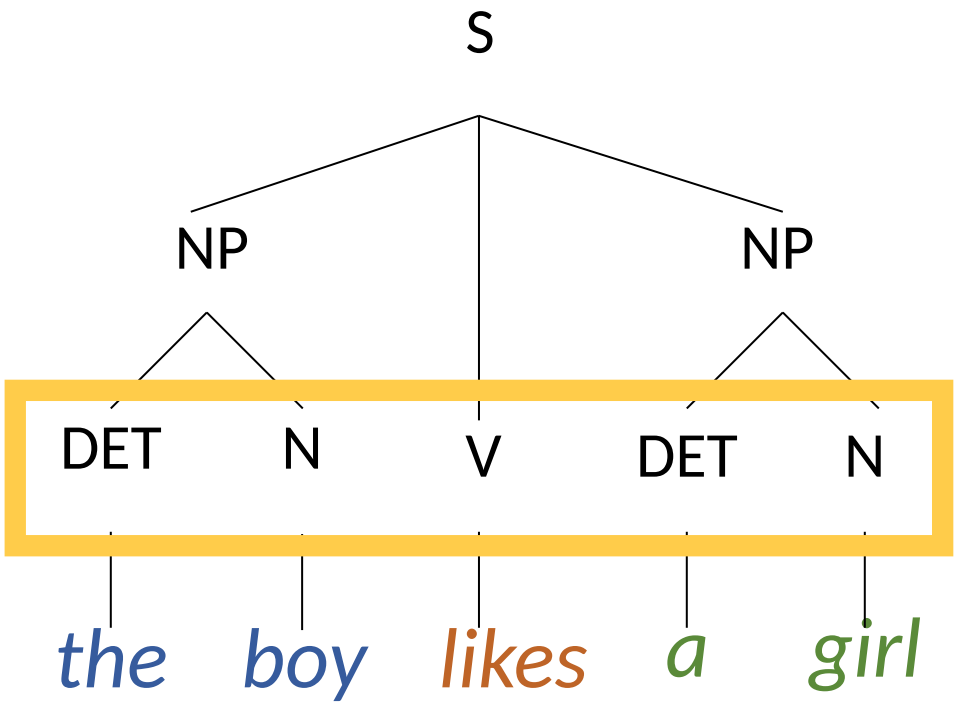
Non-terminal symbols



Constituency trees: a second attempt

- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*

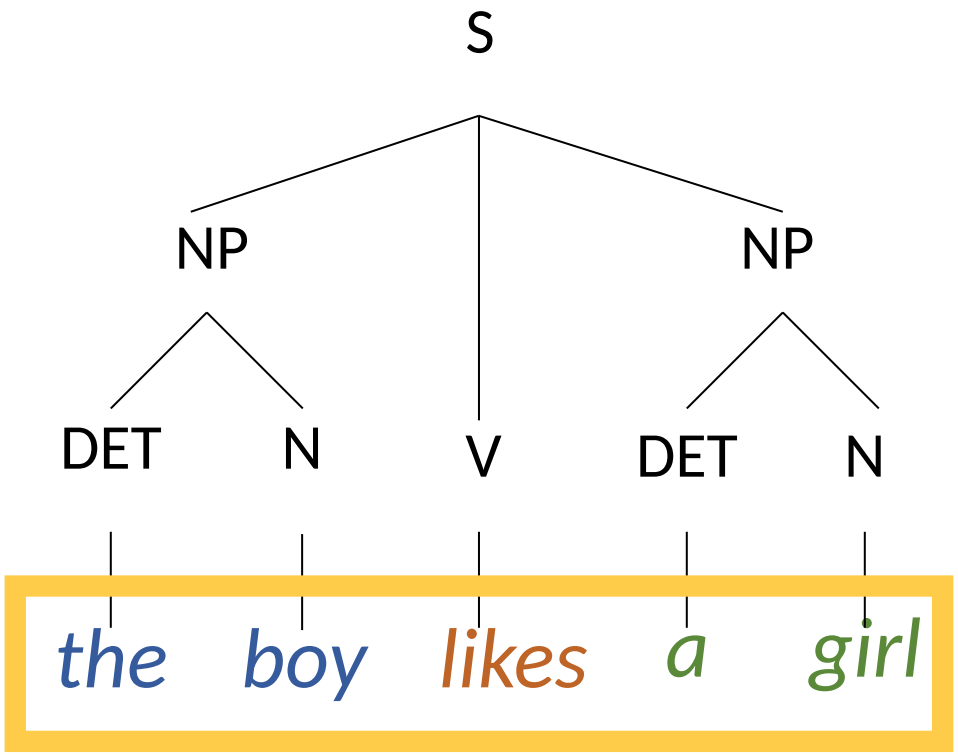
Terminal symbols



Constituency trees: a second attempt

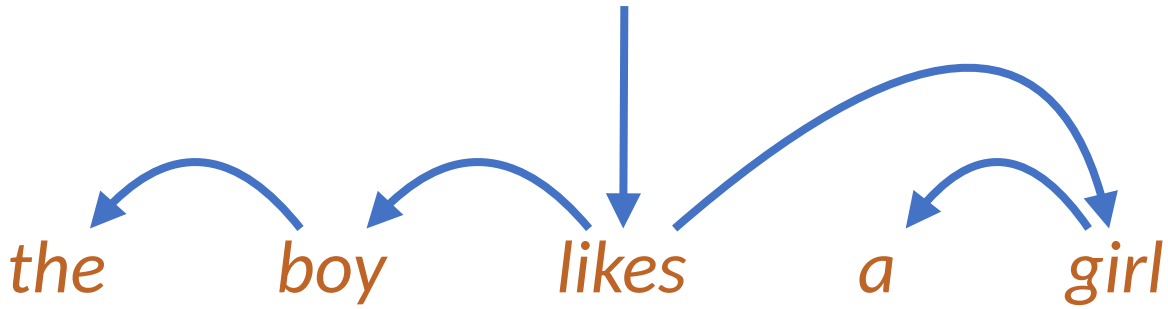
- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*

Lexical anchors



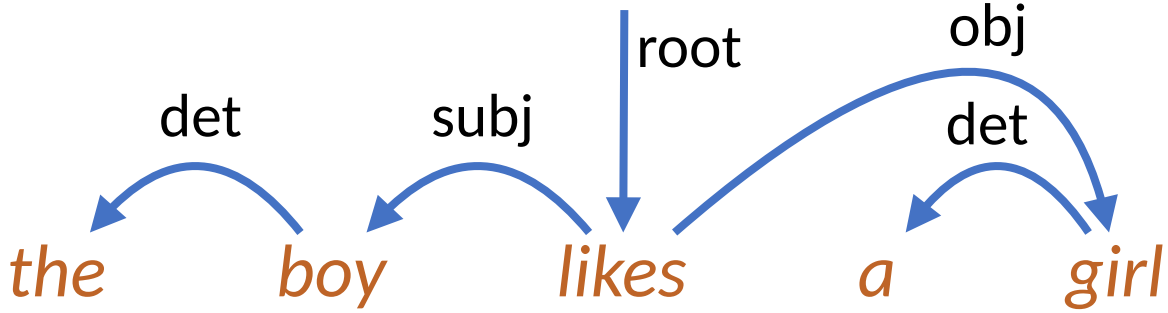
Dependencies

- Underlying idea: each word is **governed** by another word, except for the “main” word of the sentence
- A link between a word and its governor is a **dependency**



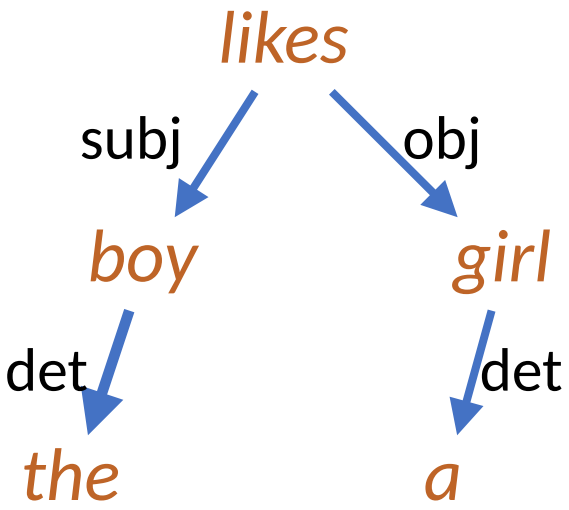
Dependencies

- Underlying idea: each word is **governed** by another word, except for the “main” word of the sentence
- A link between a word and its governor is a **dependency**
- Such links can be **labelled** with dependency types



Dependencies

- Underlying idea: each word is **governed** by another word, except for the “main” word of the sentence
- A link between a word and its governor is a **dependency**
- Such links can be **labelled** with dependency types

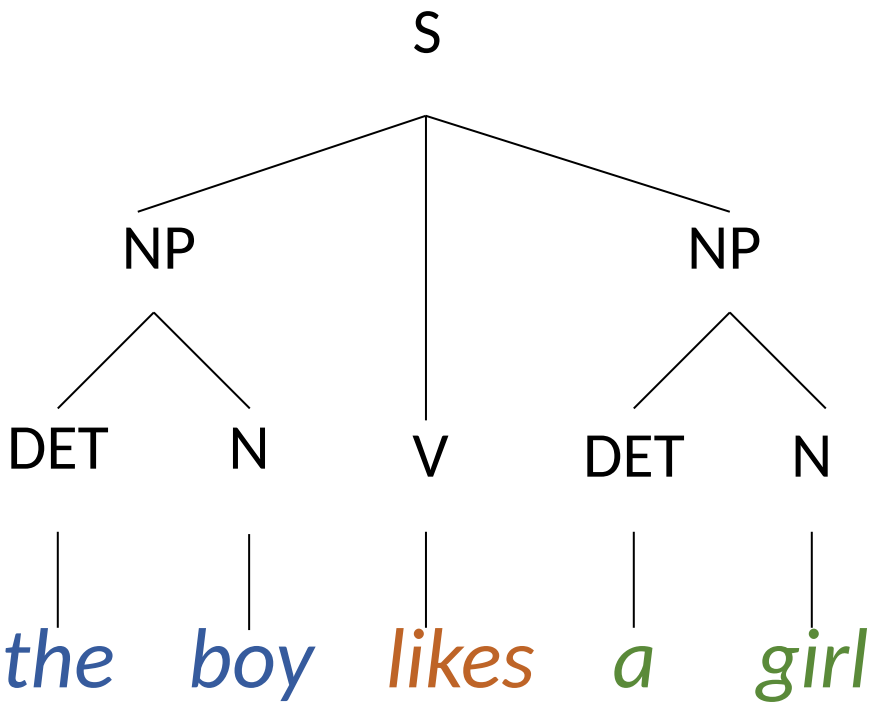


Dependencies vs. constituents

- A dependency tree provides all the information needed to create the **structure** of the constituency tree
  - But information is missing for labelling internal nodes (i.e. to know non-terminal symbols)
- A constituency tree is not enough to re-create the dependency tree
  - We need to know the **head** of each constituent
  - We have abandoned the “1 non-bracketed wordform per constituent” approximation => we need a way to know the head of each constituent

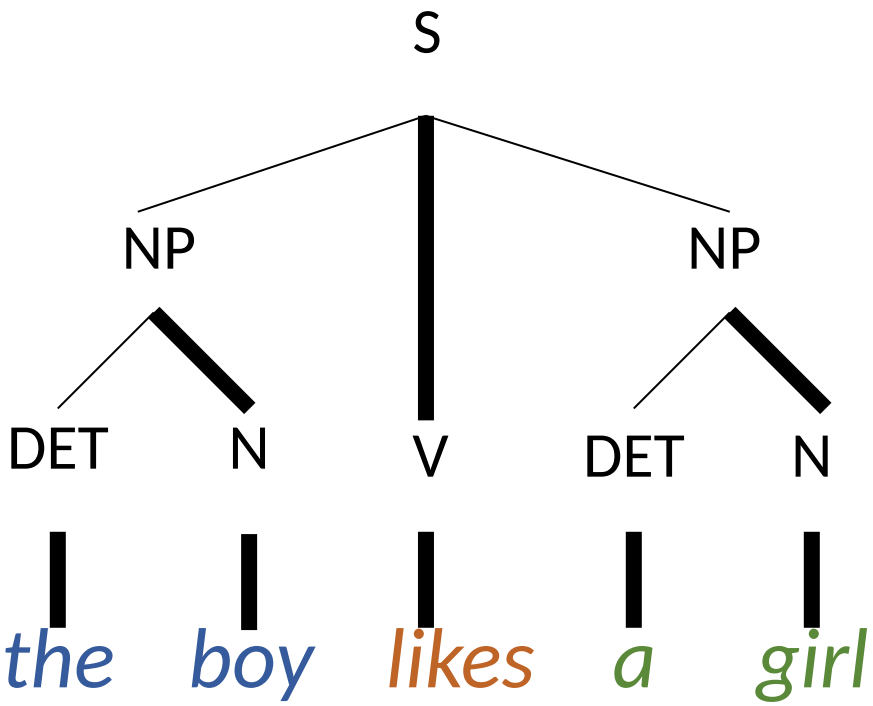
Constituency trees: specifying heads

- *(S (NP (DET the) (N boy)) (V likes) (NP (DET a) (N girl)))*



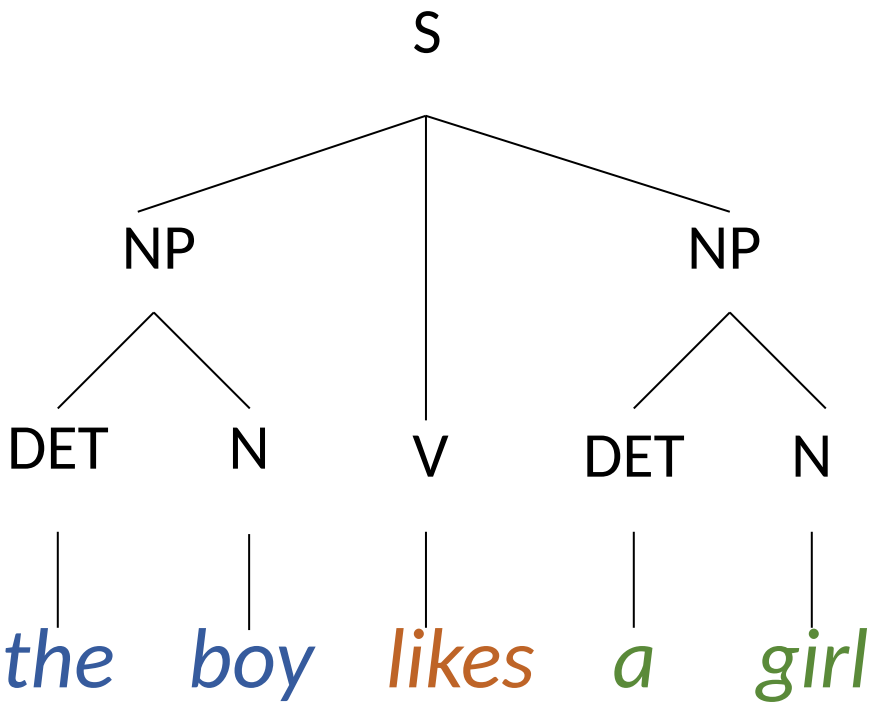
Constituency trees: specifying heads

- Explicit specification



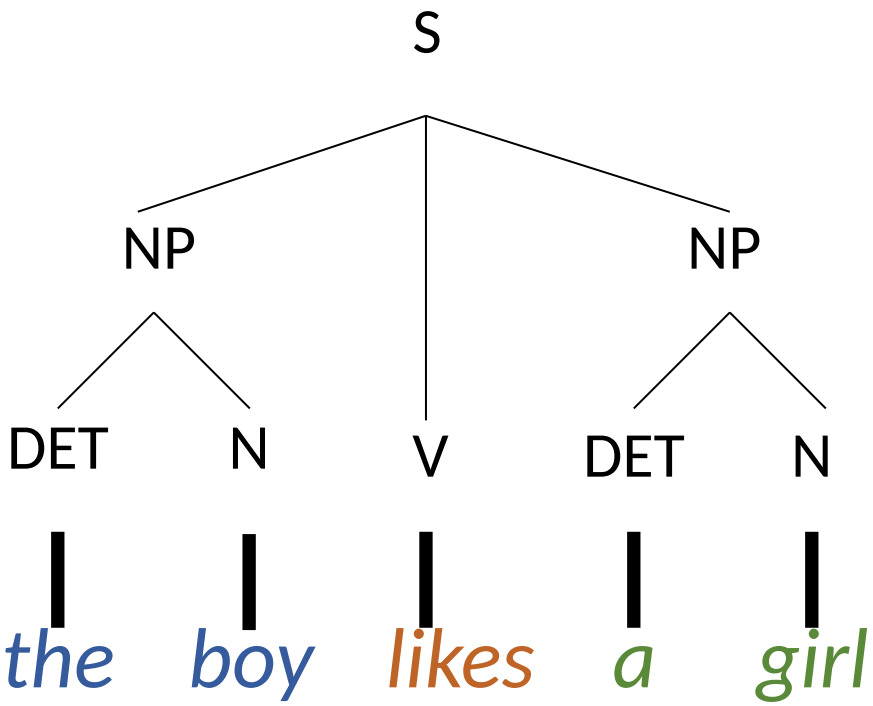
Constituency trees: specifying heads

- Head percolation table



Constituency trees: specifying heads

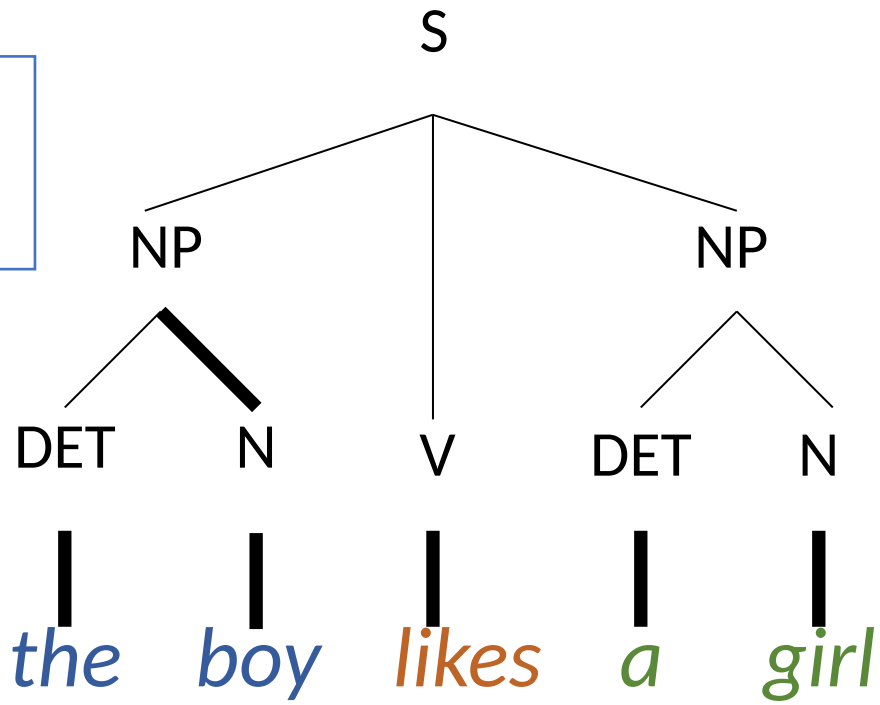
- Head percolation table



Constituency trees: specifying heads

- Head percolation table

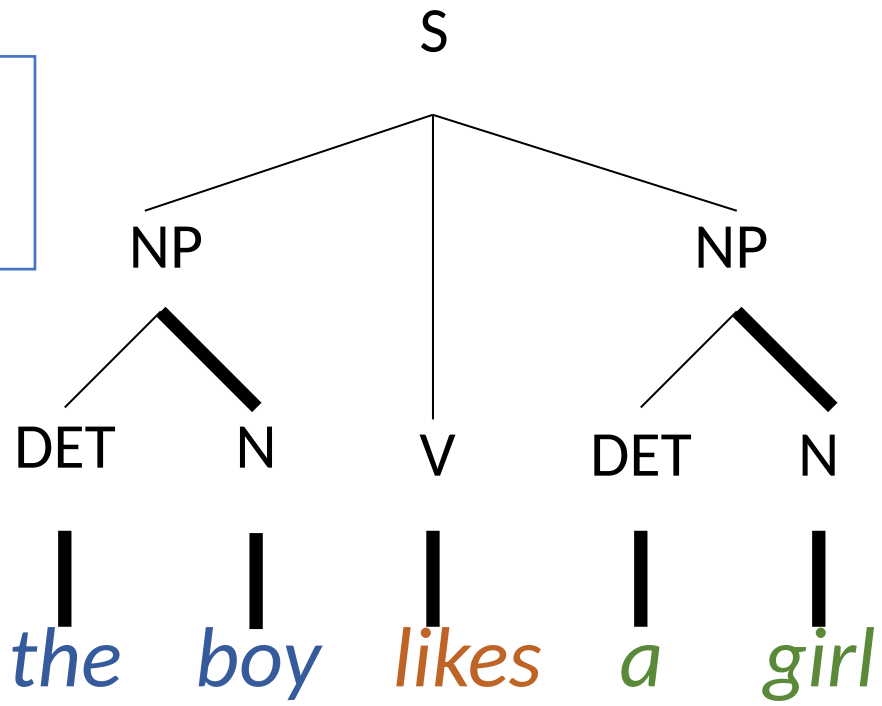
NP (DET **N***)



Constituency trees: specifying heads

- Head percolation table

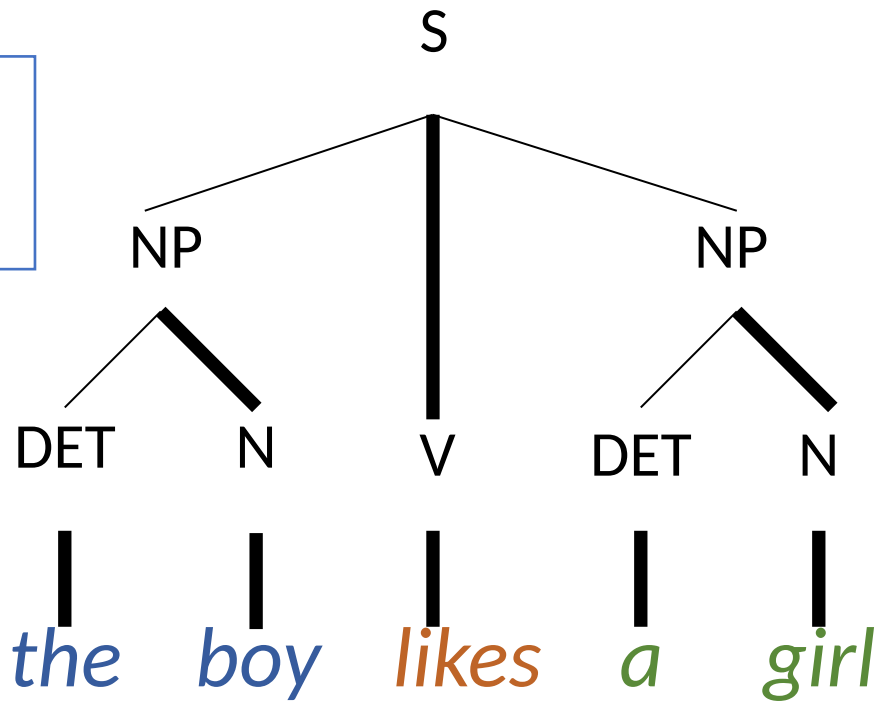
NP (DET **N***)



Constituency trees: specifying heads

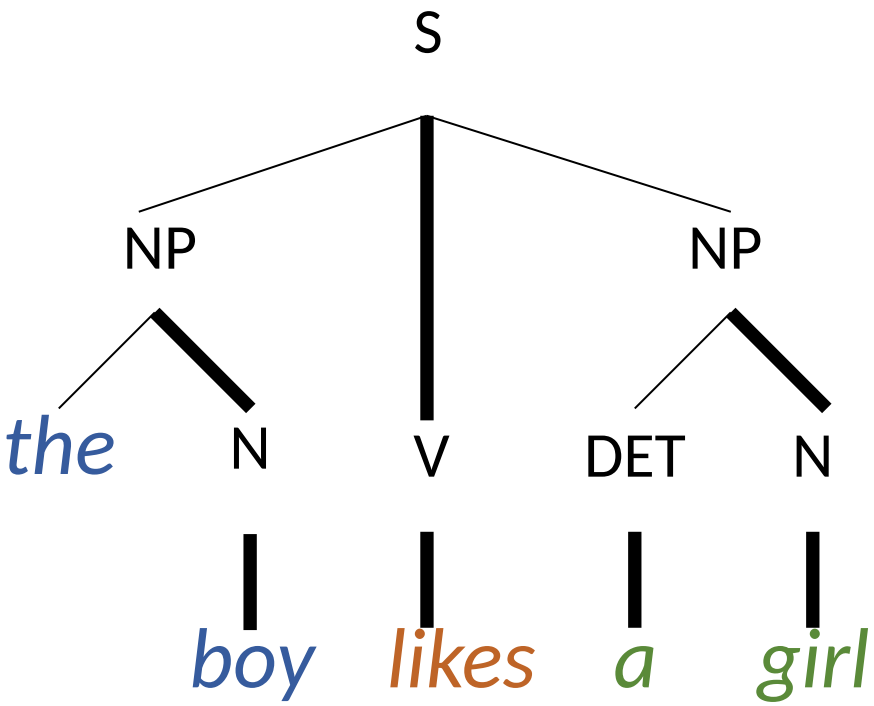
- Head percolation table

S (… **V*** …)



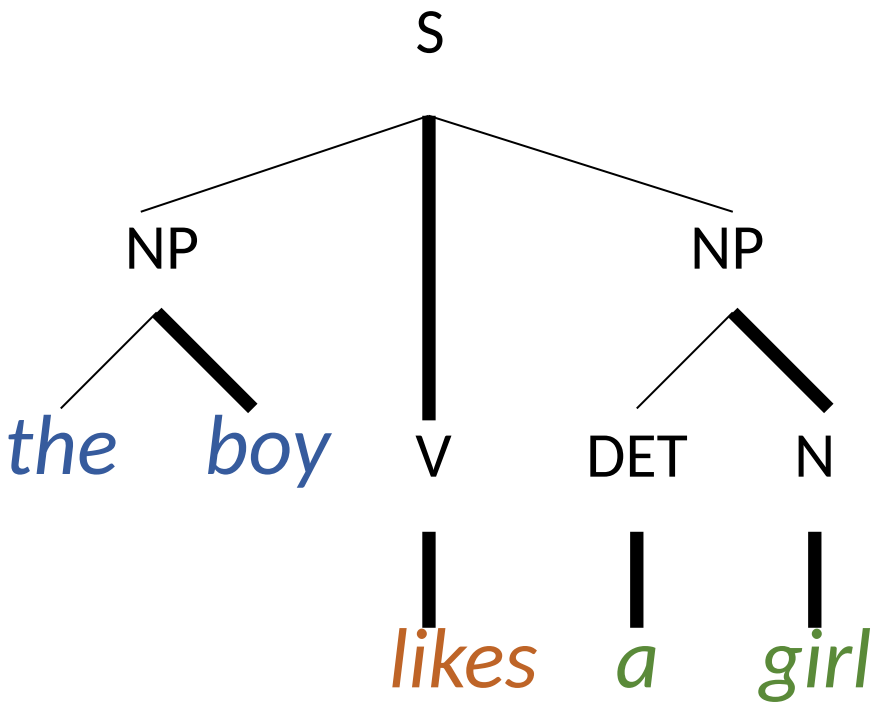
Constituency trees: specifying heads

- Head percolation table



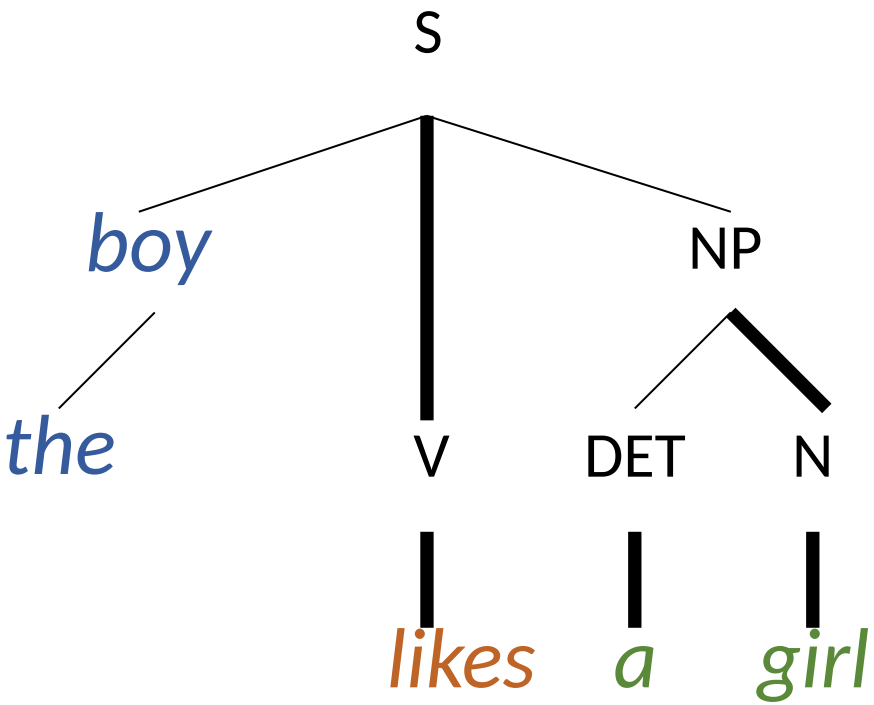
Constituency trees: specifying heads

- Head percolation table



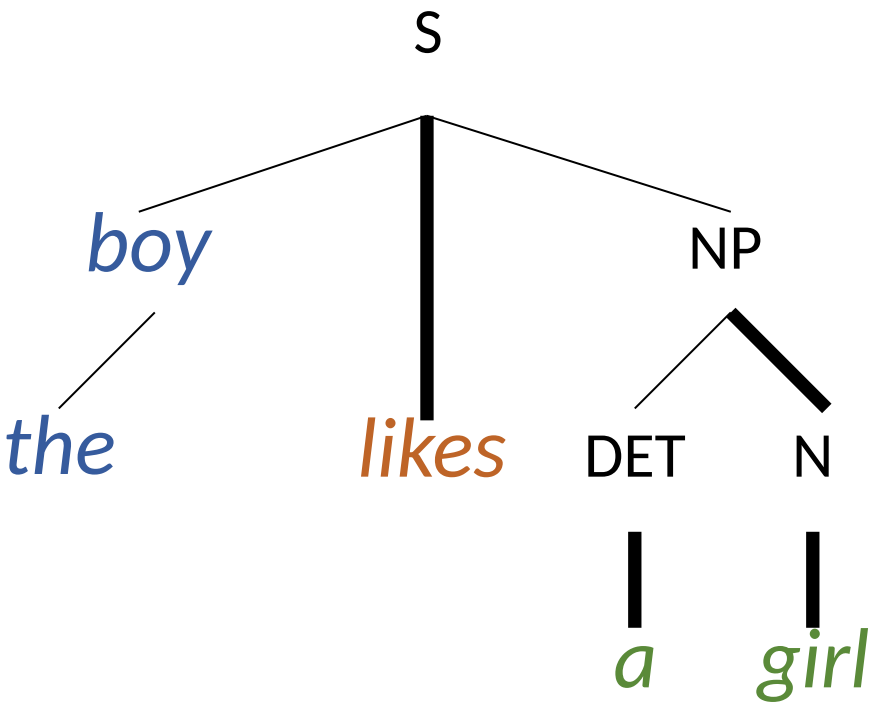
Constituency trees: specifying heads

- Head percolation table



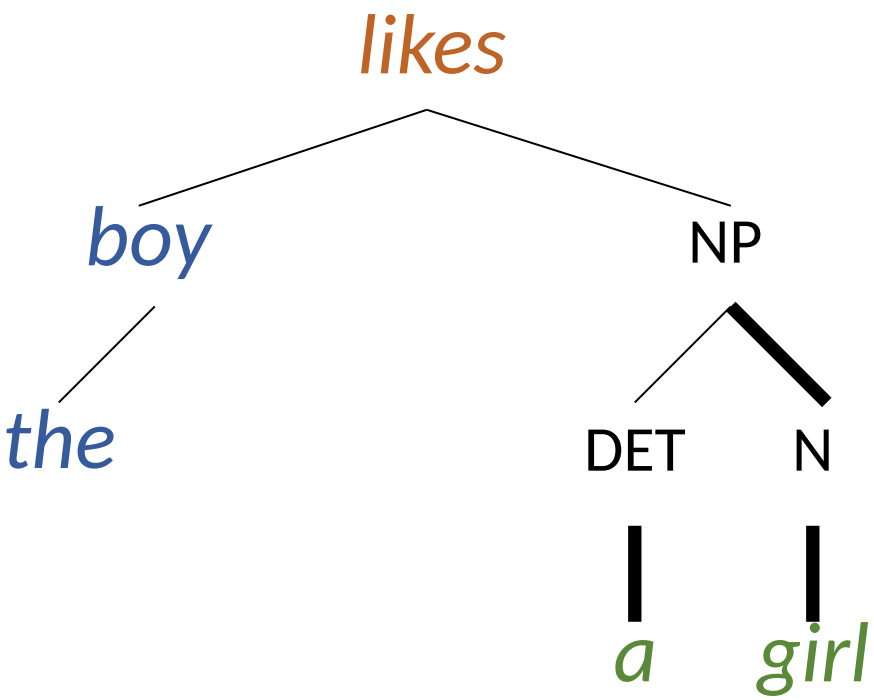
Constituency trees: specifying heads

- Head percolation table



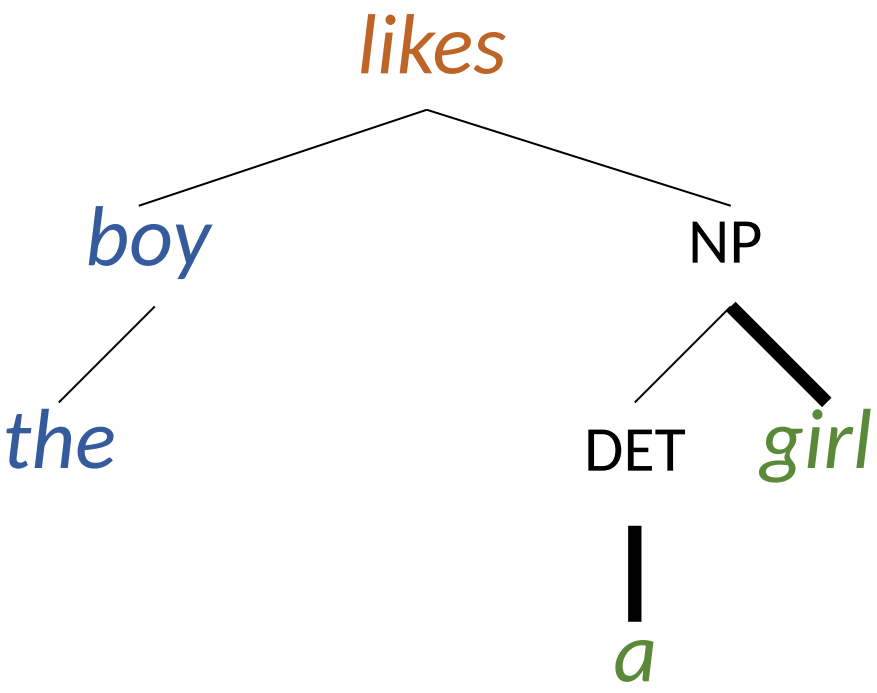
Constituency trees: specifying heads

- Head percolation table



Constituency trees: specifying heads

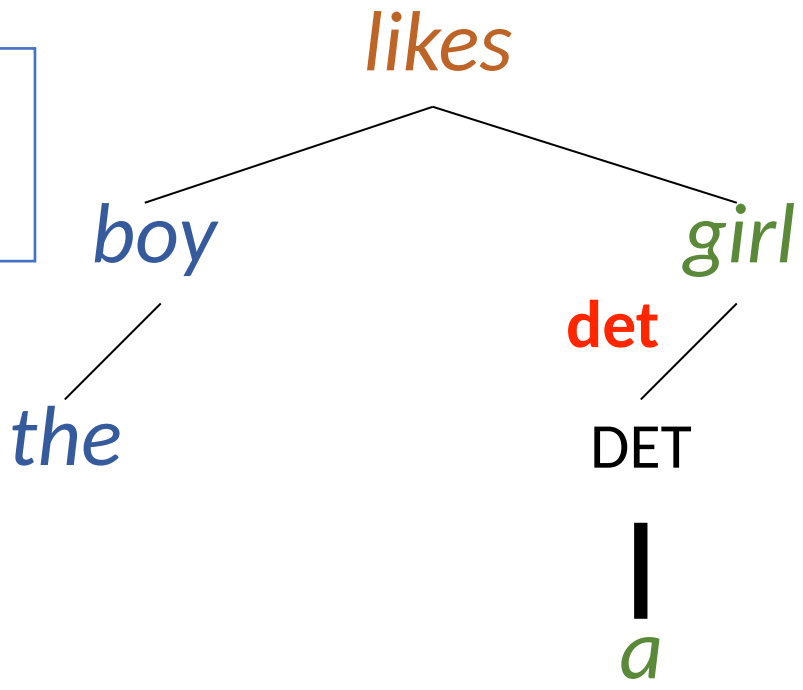
- Head percolation table



Constituency trees: specifying heads

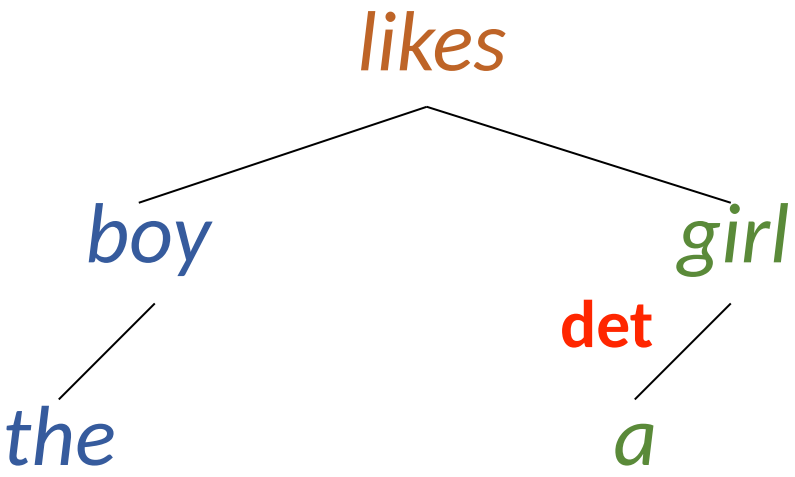
- Head percolation table, which can include dependency type information

NP (DET$^{\text{det}}$ **N**)



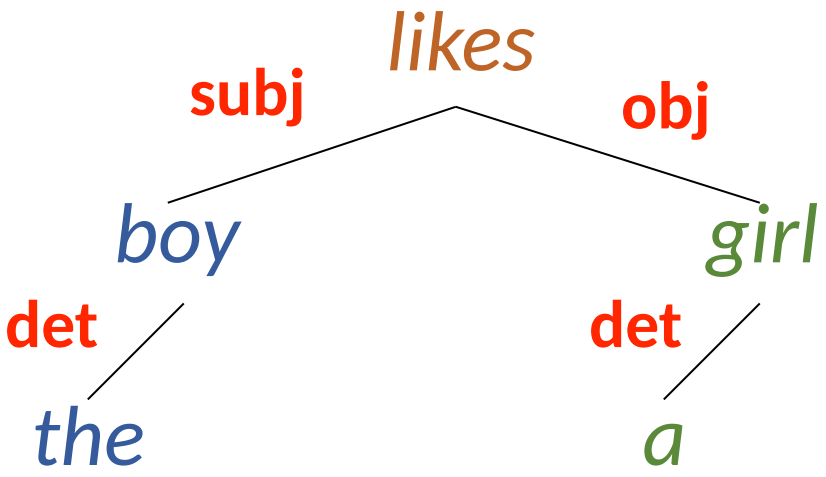
Constituency trees: specifying heads

- Head percolation table, which can include dependency type information



Constituency trees: specifying heads

- Head percolation table, which can include dependency type information

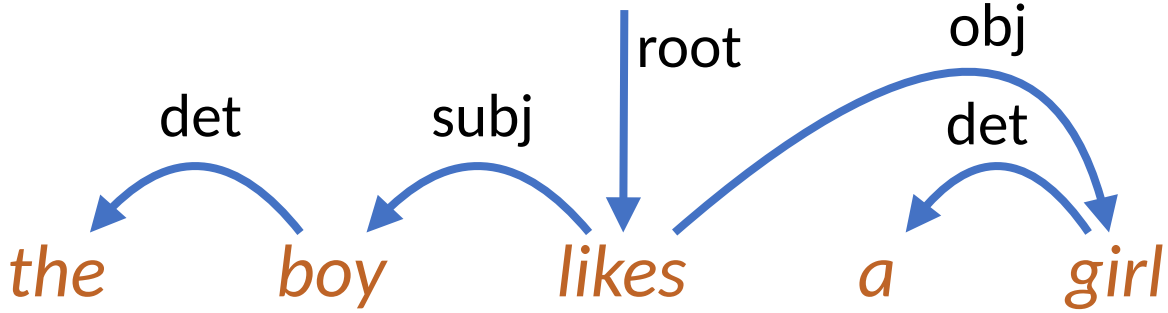


Key observation

- **Dependency structures** look like semantic structures
  - They are more useful for downstream applications
  - They tend to be more and more used, especially over the last ~10 years
- **Constituency structures** are still very important, especially for configurational languages such as English (and to a lesser extent French)
- The two types of structures capture **different aspects** of the syntactic structure of a sentence

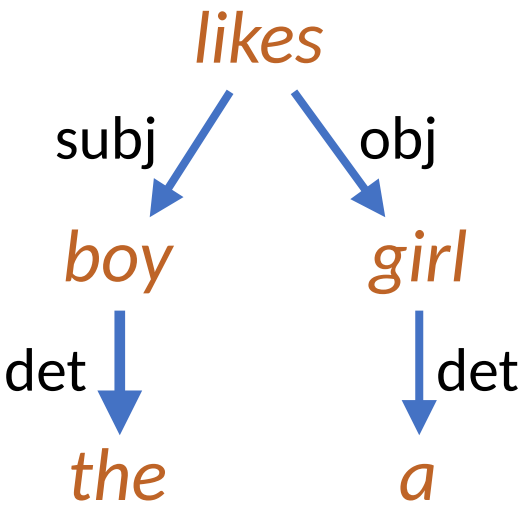
Non-projective dependencies

- **Projective case:**



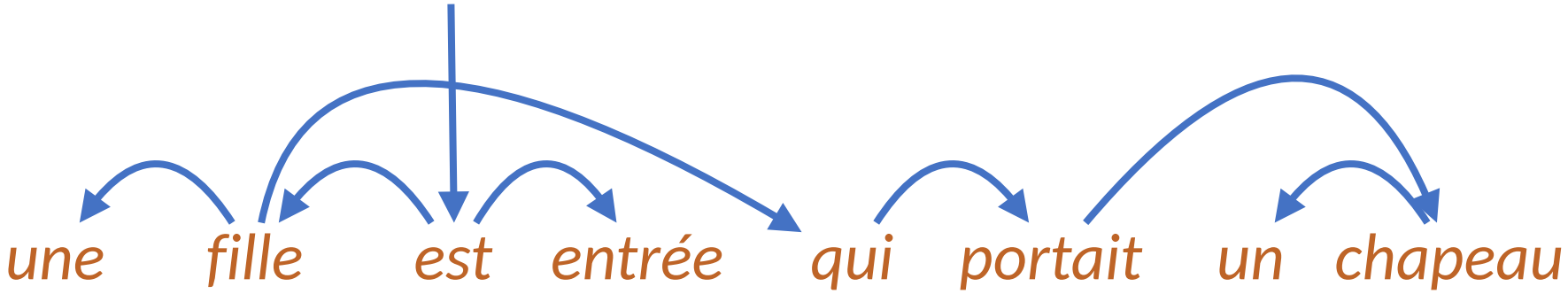
Non-projective dependencies

- **Projective case:**



Non-projective dependencies

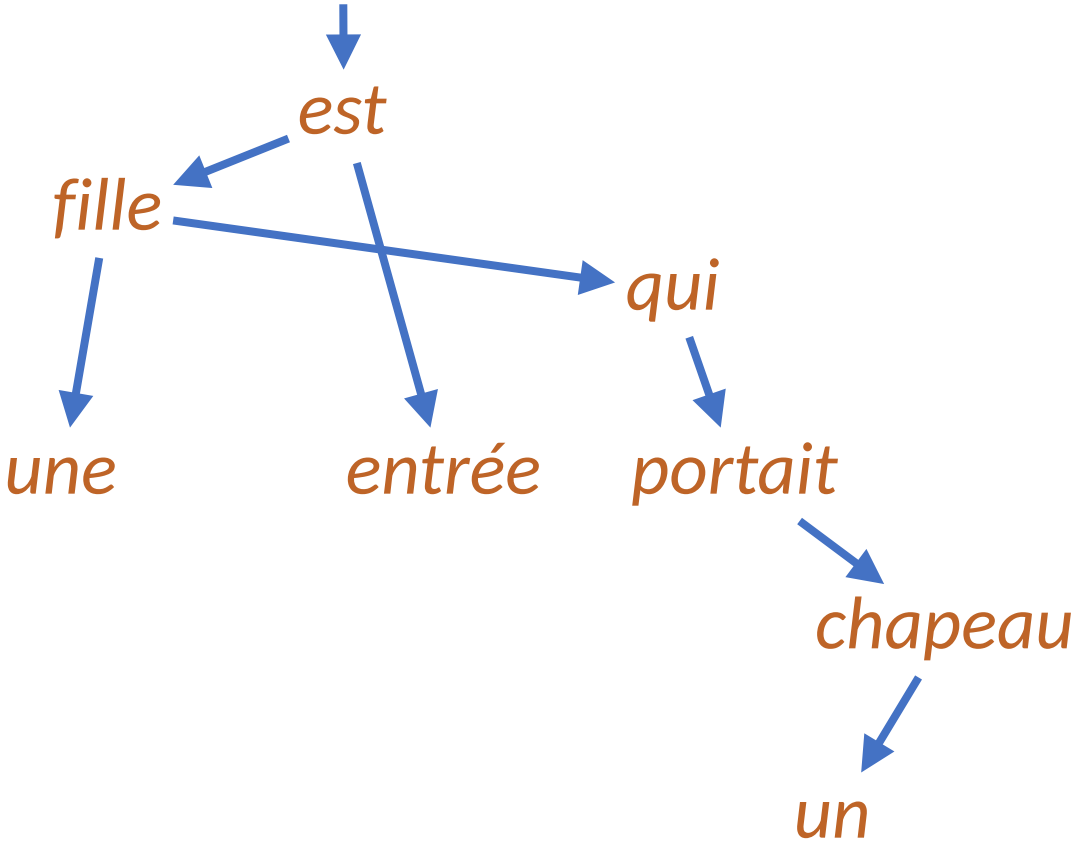
- **Non-projective case:**



‘A girl entered, who was wearing a hat’

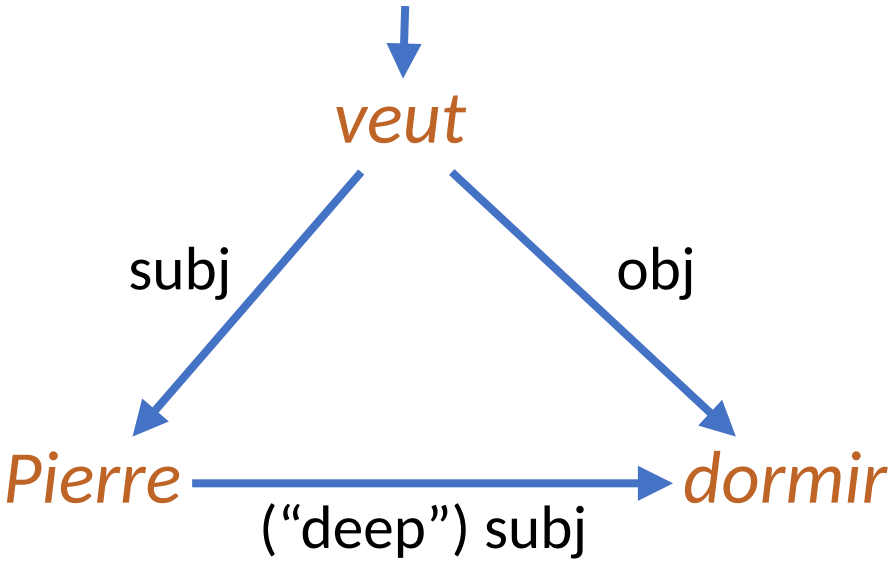
Non-projective dependencies

- **Non-projective case:**



Control, raising and attribution

- **Non-tree case:**

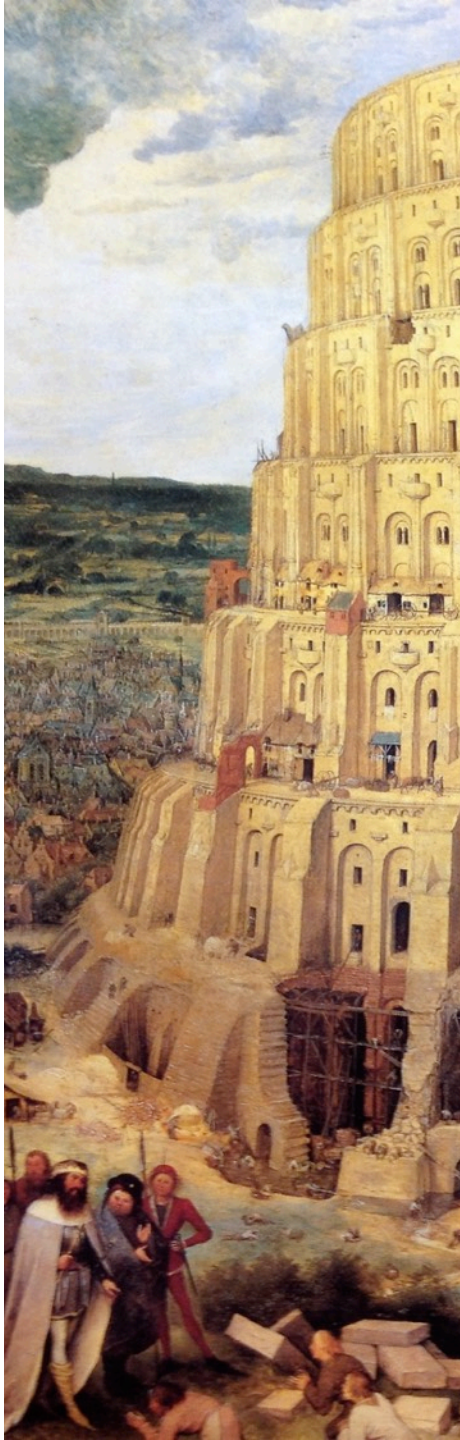


‘Peter wants to sleep’

Overall objective

- Ideally, we would like to be able to **simultaneously** generate for each sentence both:
  - a constituency structure,
  - a semantic-ish dependency-like structure
- We will limit ourselves to **projective, tree-like** structures
- We will start with constituents

Formal grammars



Language

- **Language** = an set of words over an alphabet T, called the vocabulary
- In other words, a language is a subset of T^*
  E.g. {*a*, *ab*, *aa*, *aaa*, *aab*,....}
  - Finite or infinite
  - T^* is infinite yet countable (enumerable)
  - The number of languages defined over T^* (i.e. how many subsets, i.e. the size of powerset of T^*) is non-enumerable
- Note: in NLP, we tend to replace “word” (element of the language) with “sentence”, and “character” (element of T) with “word”
- More examples:

  $L_1 = \{a, b\}$

  $L_2 = \{a, b, \varepsilon\}$

  $L_3 = \{a^n b^n \mid n \in \mathbb{N}\}$

  $L_4 = \{ww^{-1} \mid w \in T\}$, where w^{-1} is the mirror image of w

Grammar

A grammar G is defined as a quadruple

$$G = (V, T, S, P)$$

where

V is a finite set of objects called **variables**, or **non-terminal symbols**,

T is a finite set of objects called **terminal symbols**,

$S \in V$ is a special symbol called the **start** variable,

P is a finite set of **productions**, or **rewriting rules**

Grammar: an example

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \to aSb,$$
$$S \to \varepsilon .$$

Then

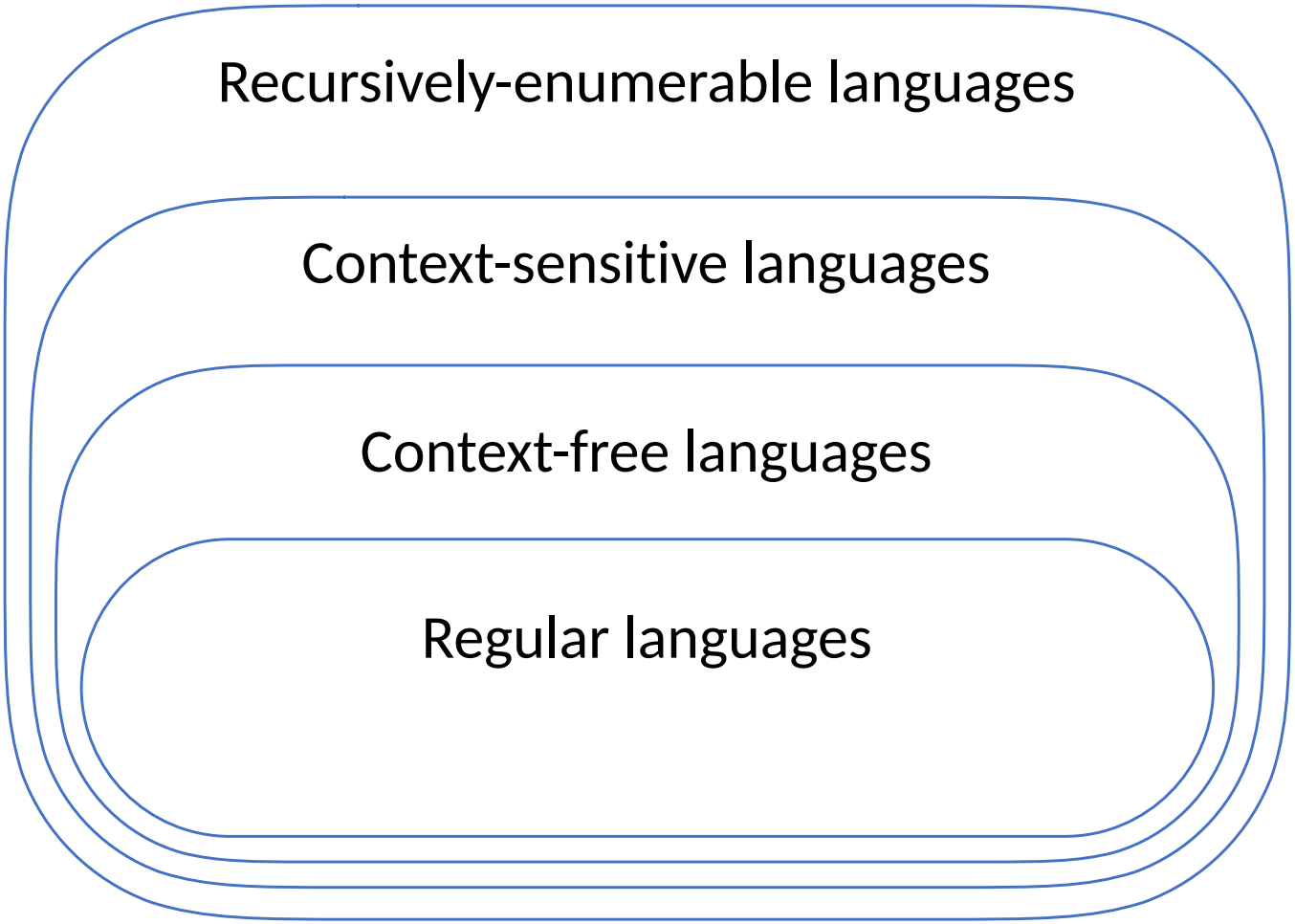
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

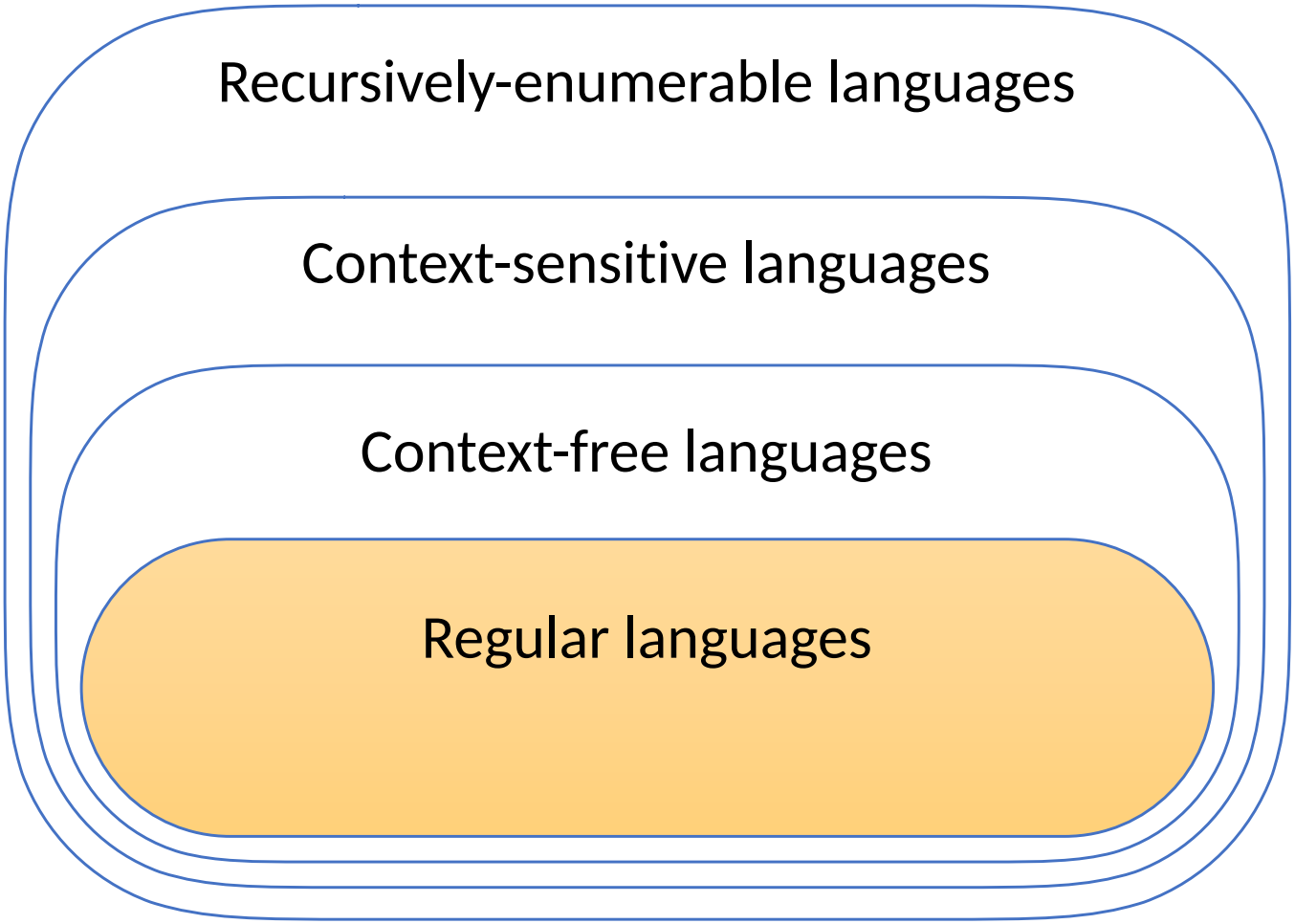
$$S \overset{*}{\Rightarrow} aabb.$$

The string *aabb* is a sentence in the language generated by G, while *aaSbb* is a sentential form.

Chomsky hierarchy



Chomsky hierarchy

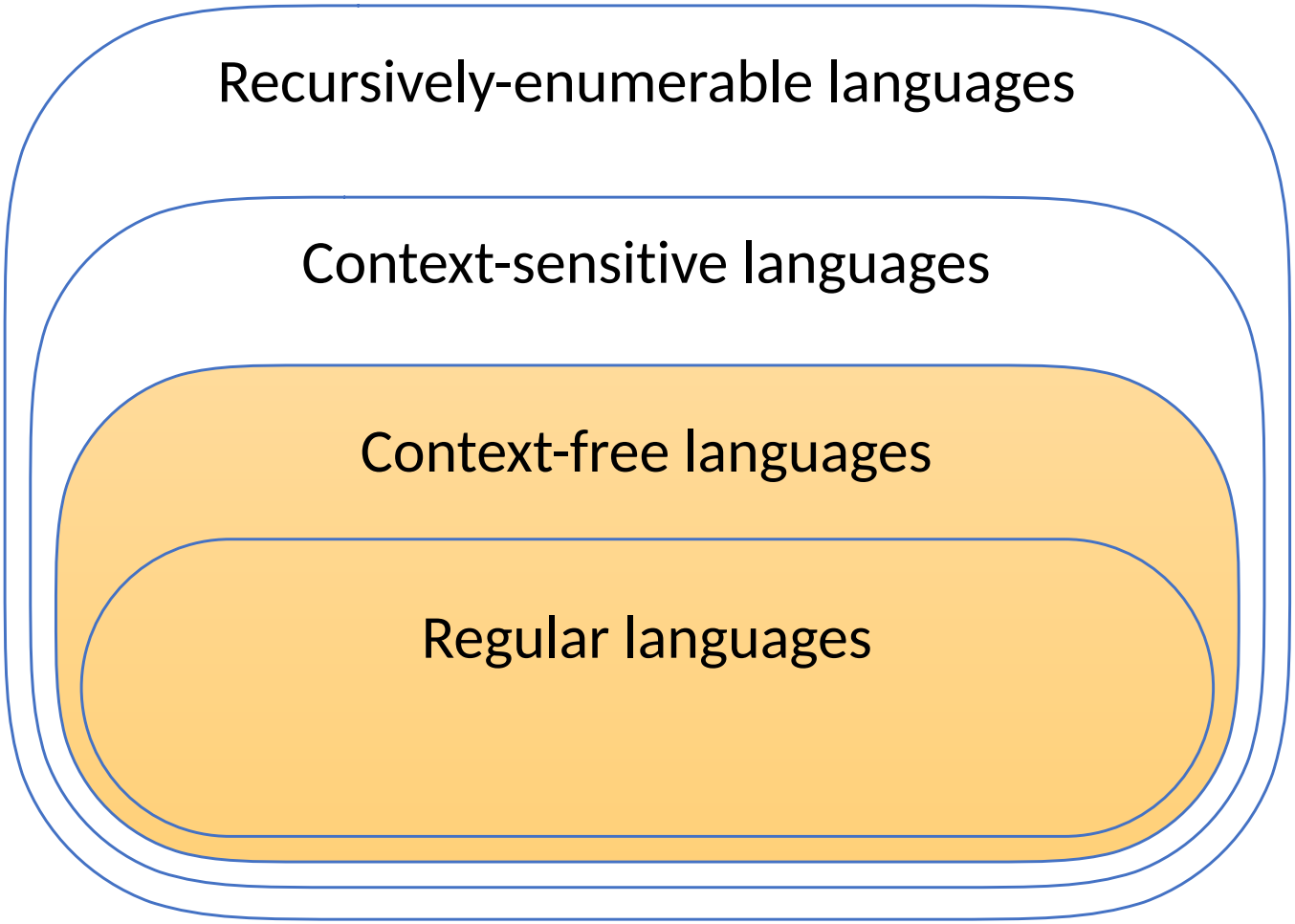


Allowed rewriting rule forms:
- B -> a
- B -> aC
- B -> ε

OR
- B -> a
- B -> Ca
- B -> ε

Complexity:
O(*n*)

Chomsky hierarchy

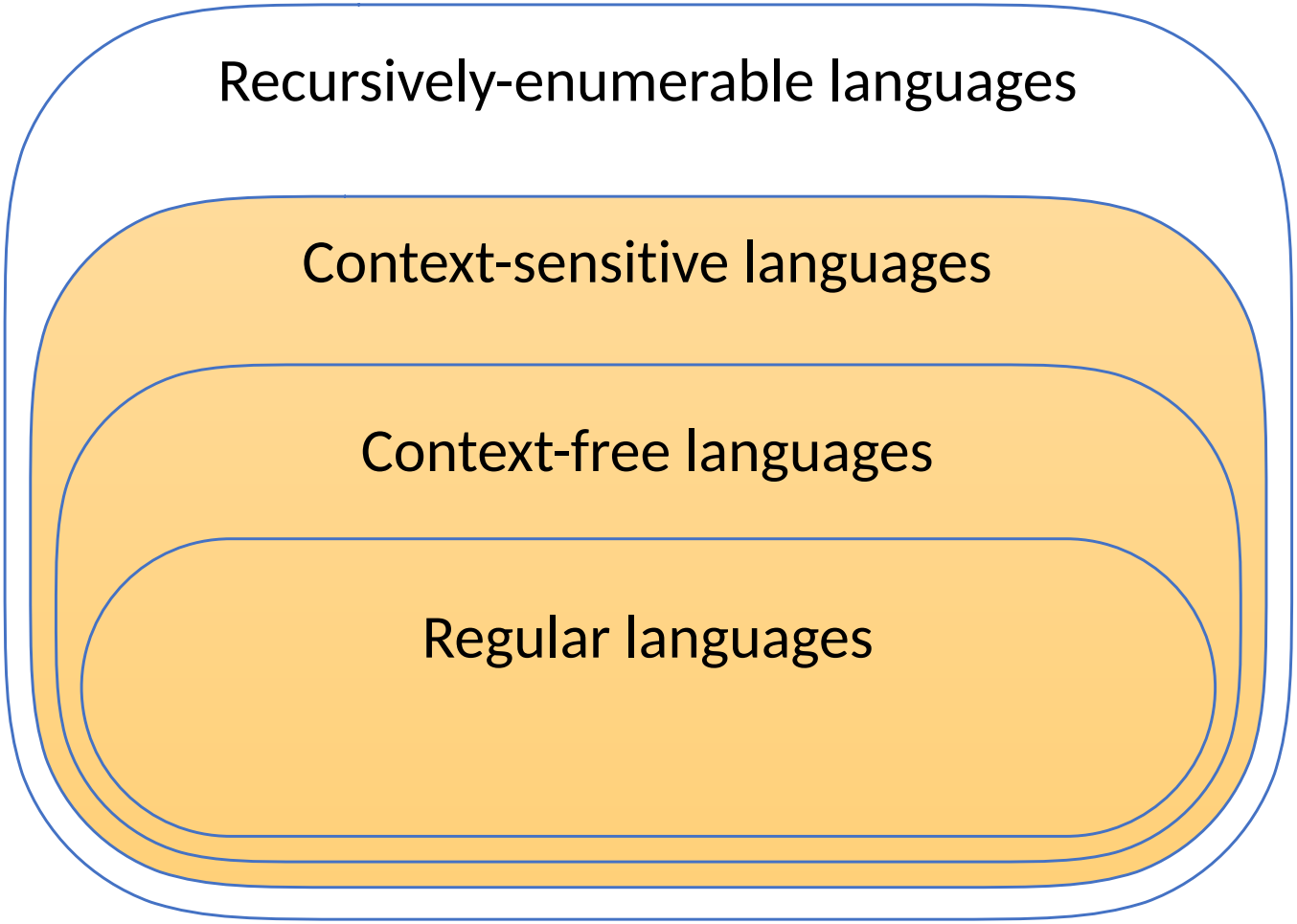


Allowed rewriting rule forms:
- B -> γ

where γ is zero, one or more terminal and/or non-terminal symbols

Complexity:
$O(n^3)$

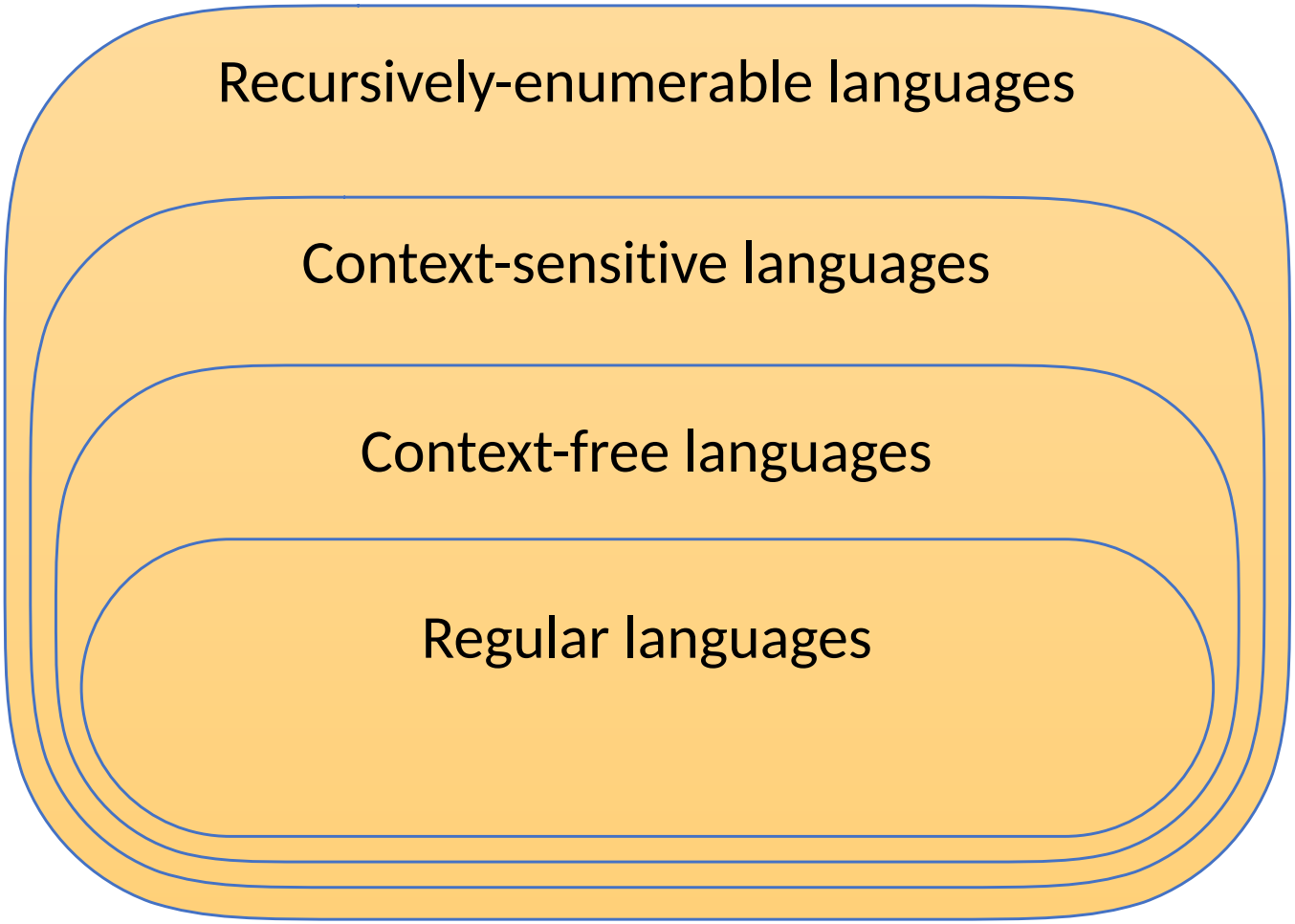
Chomsky hierarchy



Allowed rewriting rule forms:
- αBβ -> αγβ where α, β and γ are zero, one or more terminal and/or non-terminal symbols

Complexity:
NP-complete

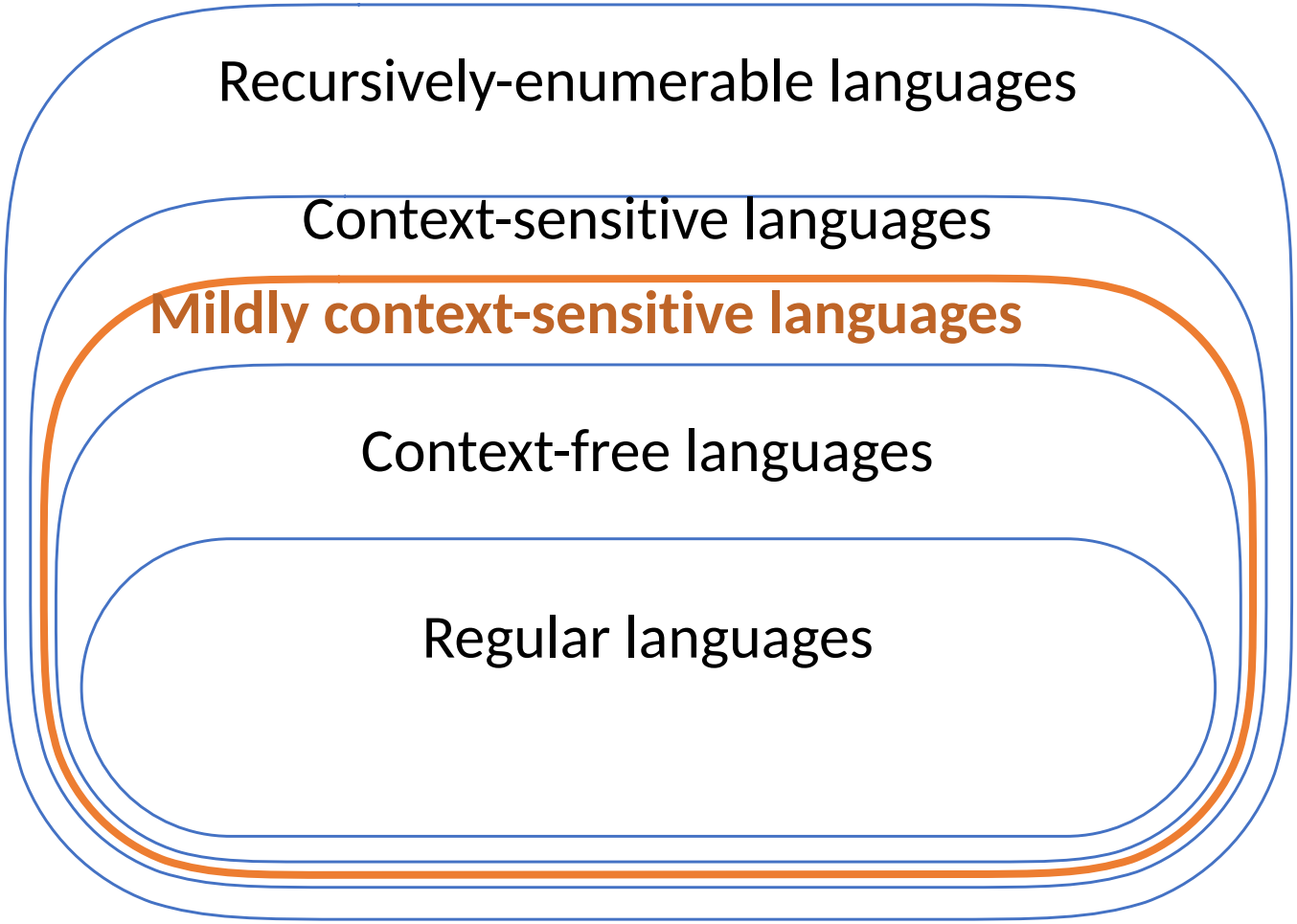
Chomsky hierarchy



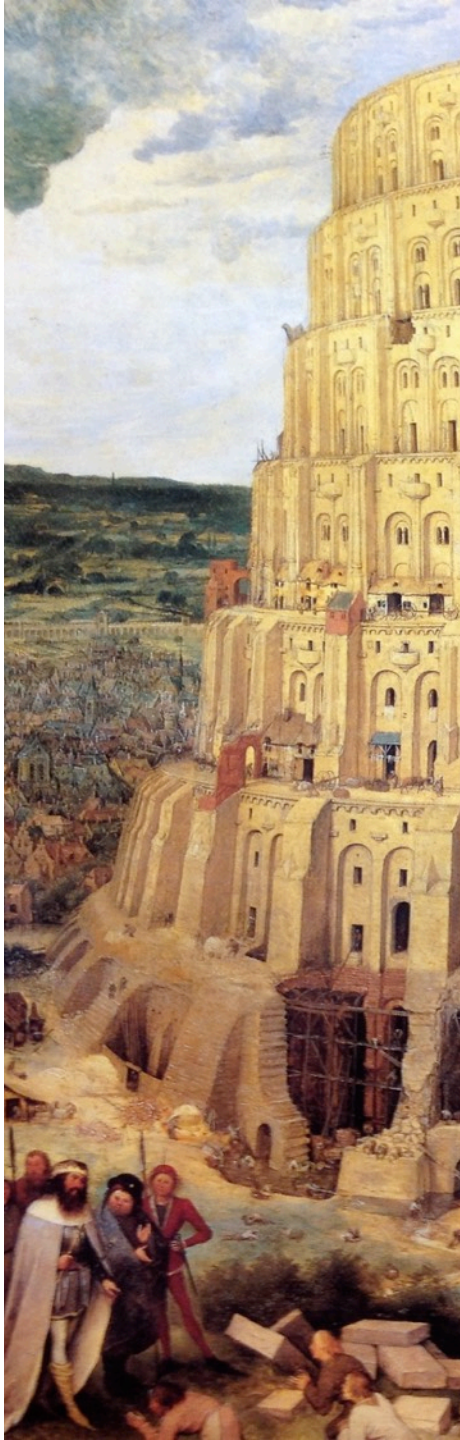
Allowed rewriting rule forms:
- α -> β
where α and β are the same as before

Complexity:
Undecidable (=halting problem for Turing machines)

Chomsky hierarchy



Context-free grammars



Context-free grammars (CFGs)

- (Note to French speakers: context-free grammars, grammaires hors-contexte, grammaires non-contextuelles, grammaires algébriques)

Context-free grammars (CFGs)

- From now on, terminals (PoS) are lowercase, non-terminals are uppercase, and the left-hand side of the first rule of a grammar is its axiom
- Example grammar:

1 S ⟶ NP VP

2 VP ⟶ v adv NP

3 NP ⟶ np

4 NP ⟶ det N

5 N ⟶ nc PP

6 PP ⟶ p nc

7 np ⟶ ‘Pierre’

8 v ⟶ ‘mange’

9 adv ⟶ ‘souvent’

10 det ⟶ ‘des’

10 nc ⟶ ‘pommes’

11 v ⟶ ‘pommes’

12 p ⟶ ‘de’

13 nc ⟶ ‘terre’

Lexicon

- As discussed above, words are associated with a PoS
- In the trees we saw earlier, PoS were terminals (there are too many words, using words as terminals would not be practical)
- We extract rules number 7+ from the grammar and store them in the form of a **lexicon**

1	S ⟶ NP VP
2	VP ⟶ v adv NP
3	NP ⟶ np
4	NP ⟶ det N
5	N ⟶ nc PP
6	PP ⟶ p nc

Pierre	np
mange	v
souvent	adv
des	det
pommes	nc
pommes	v
de	p
terre	nc

First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP

2 VP ⟶ v adv NP

3 NP ⟶ np

4 NP ⟶ det N

5 N ⟶ nc PP

6 PP ⟶ p nc

Pierre mange souvent des pommes de terre

First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP

2 VP ⟶ v adv NP

3 NP ⟶ np

4 NP ⟶ det N

5 N ⟶ nc PP

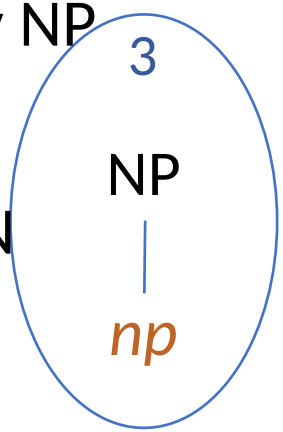
6 PP ⟶ p nc

np v adv det nc p nc

First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc

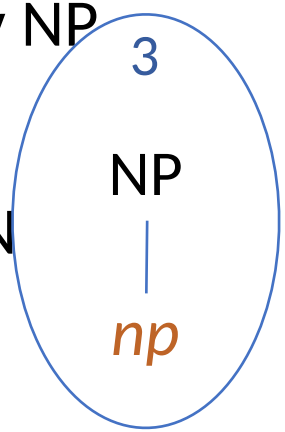


v *adv* *det* *nc* *p* *nc*

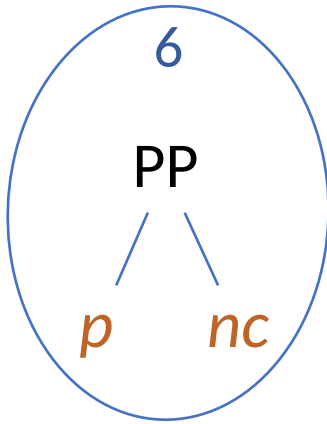
First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc



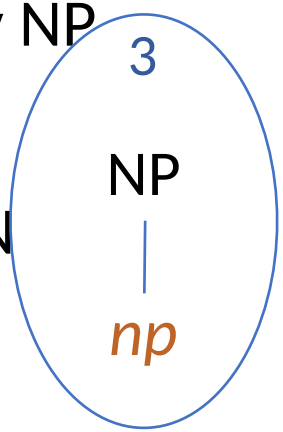
v *adv* *det* *nc*



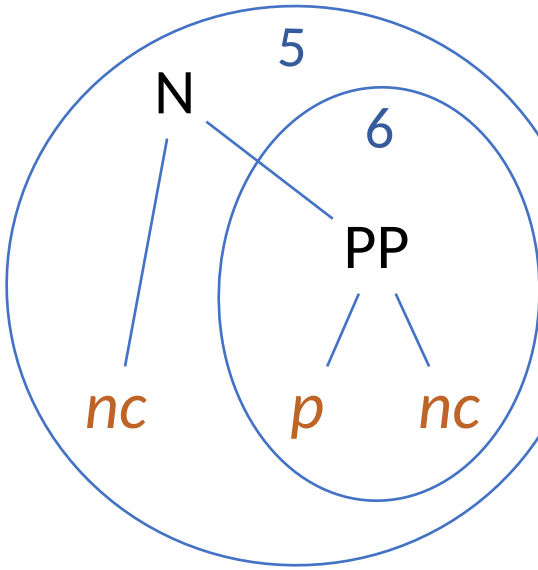
First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc



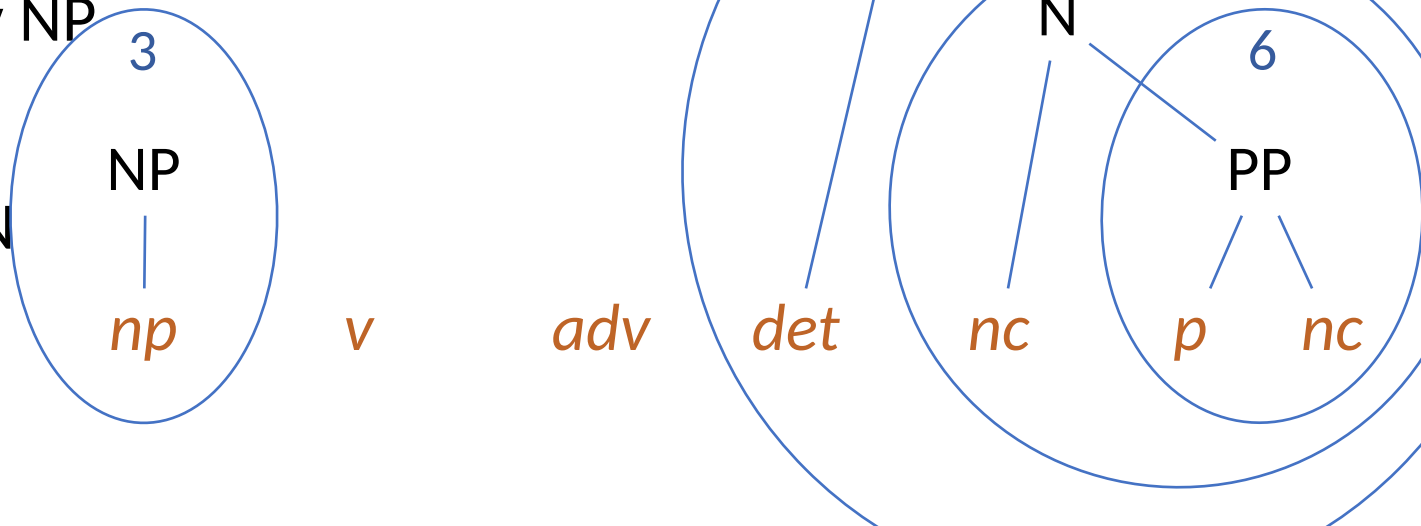
v *adv* *det*



First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

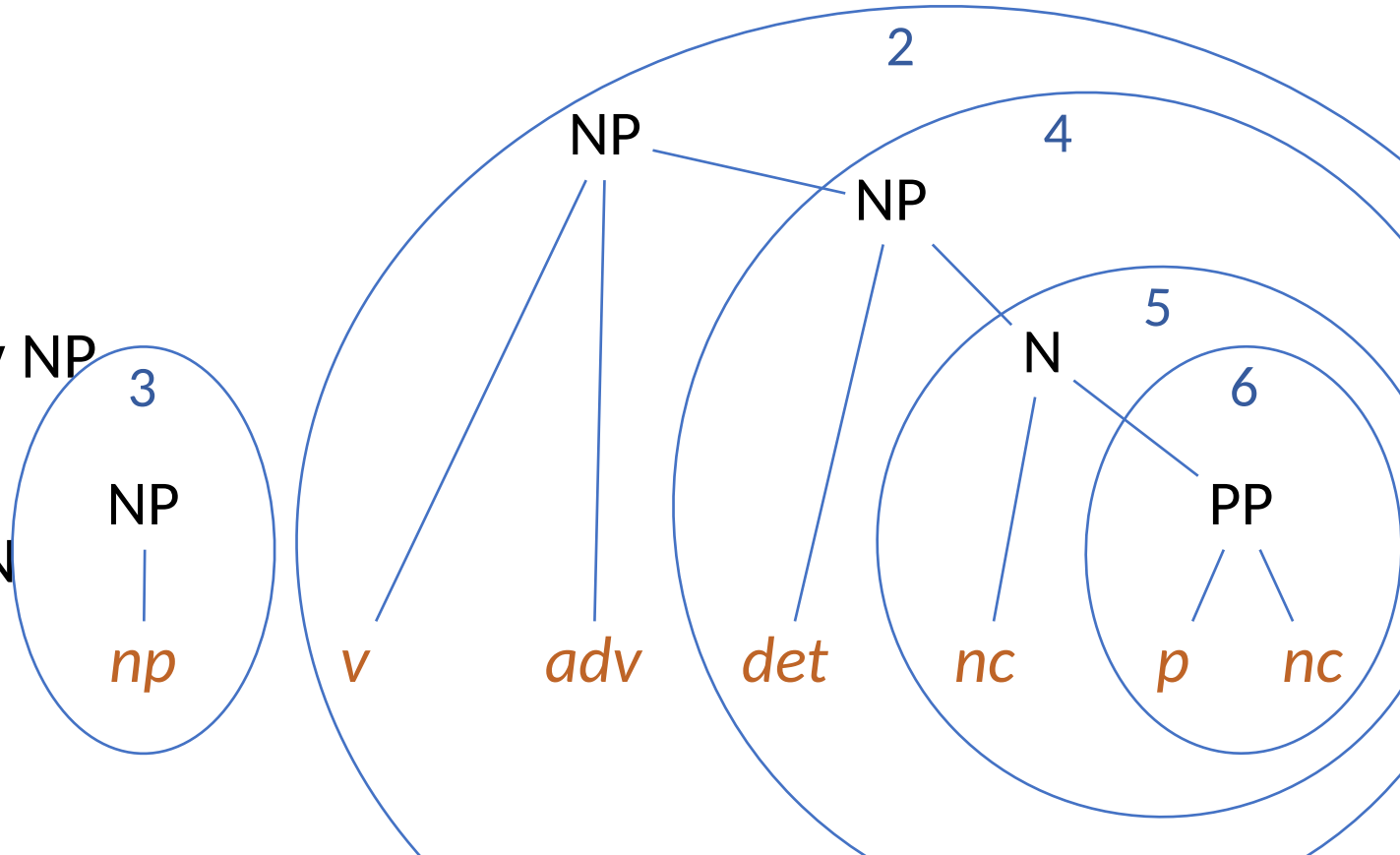
1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc



First parse

- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

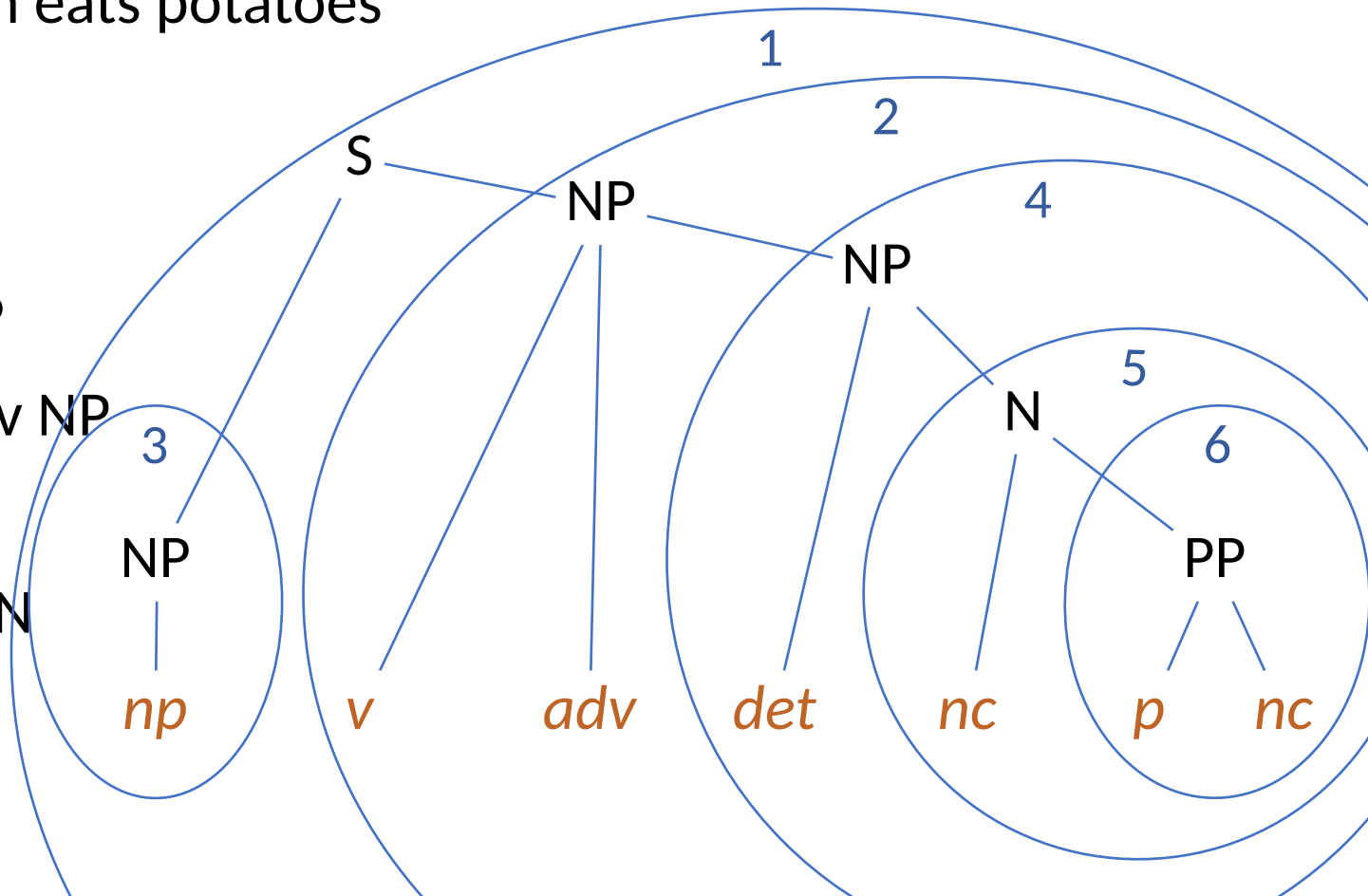
1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc



First parse

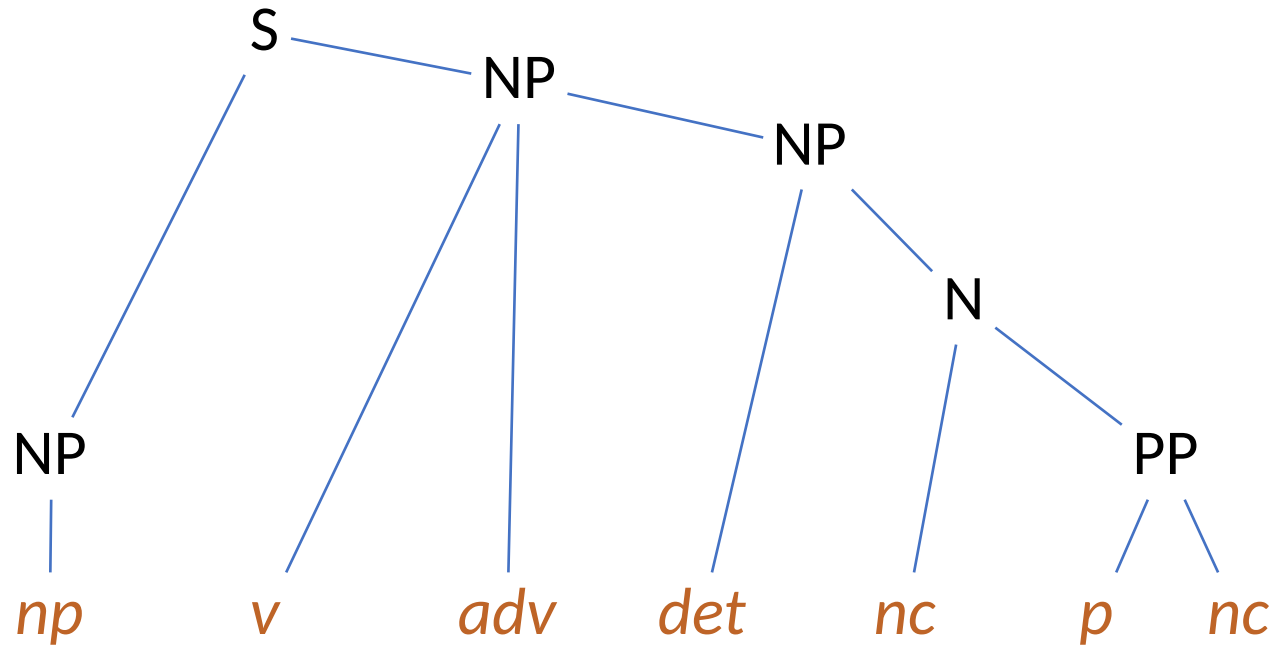
- *Pierre mange souvent des pommes de terre*
  'Peter often eats potatoes'

1 S ⟶ NP VP
2 VP ⟶ v adv NP
3 NP ⟶ np
4 NP ⟶ det N
5 N ⟶ nc PP
6 PP ⟶ p nc



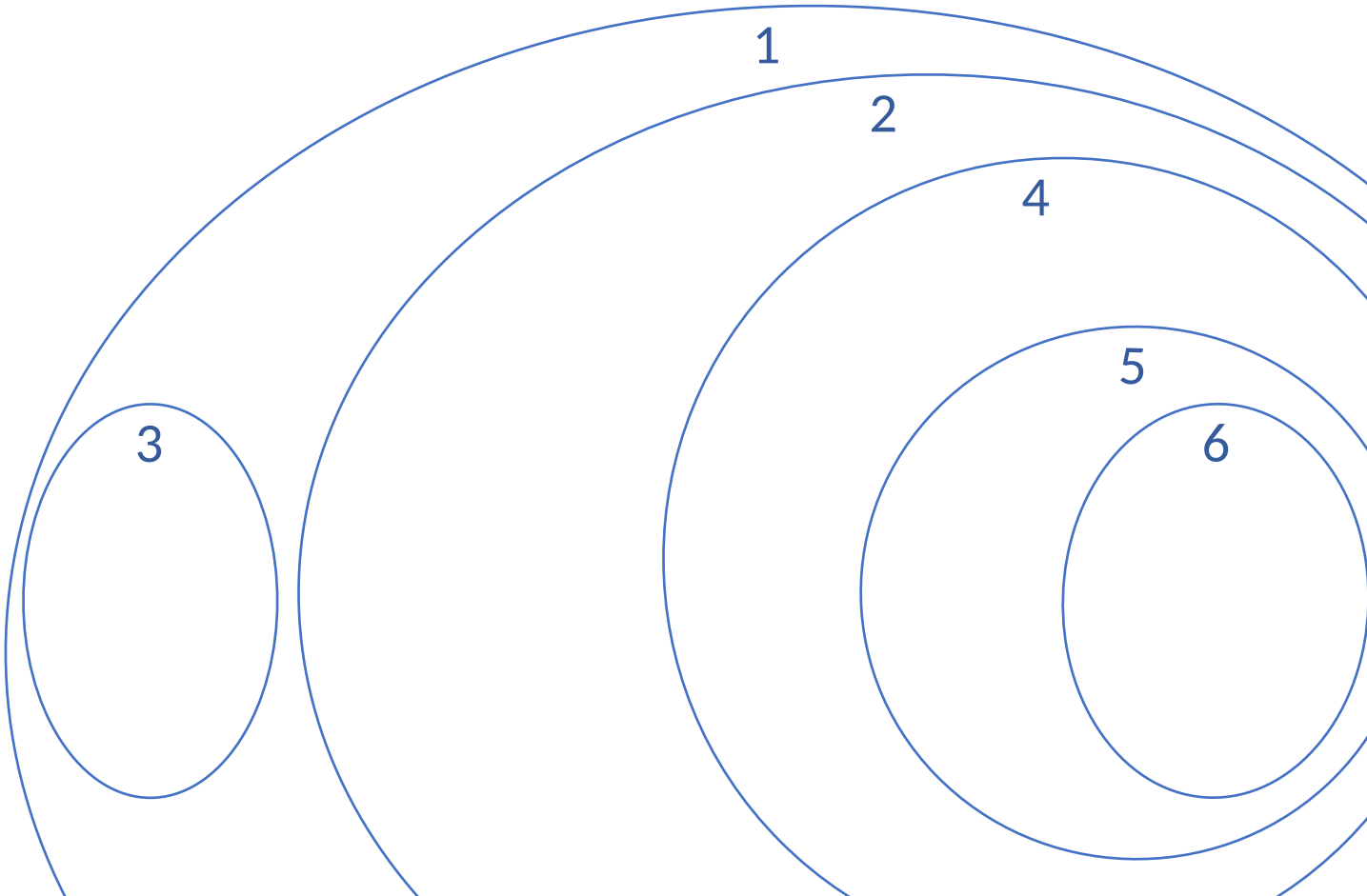
The derived tree

- **Derived tree** = outcome of the parse derivation process



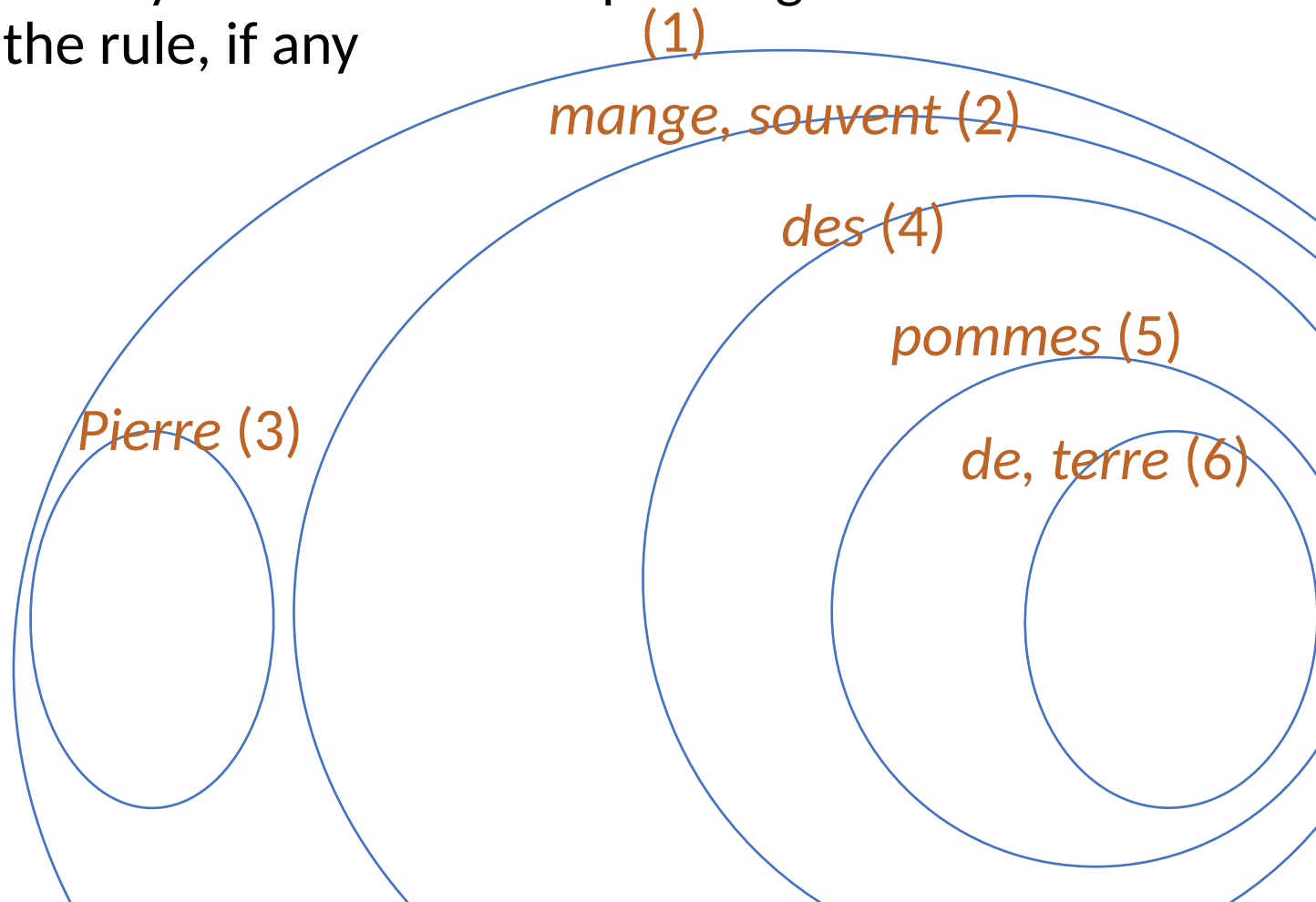
The derivation tree

- **Derivation tree = history of the derivation**



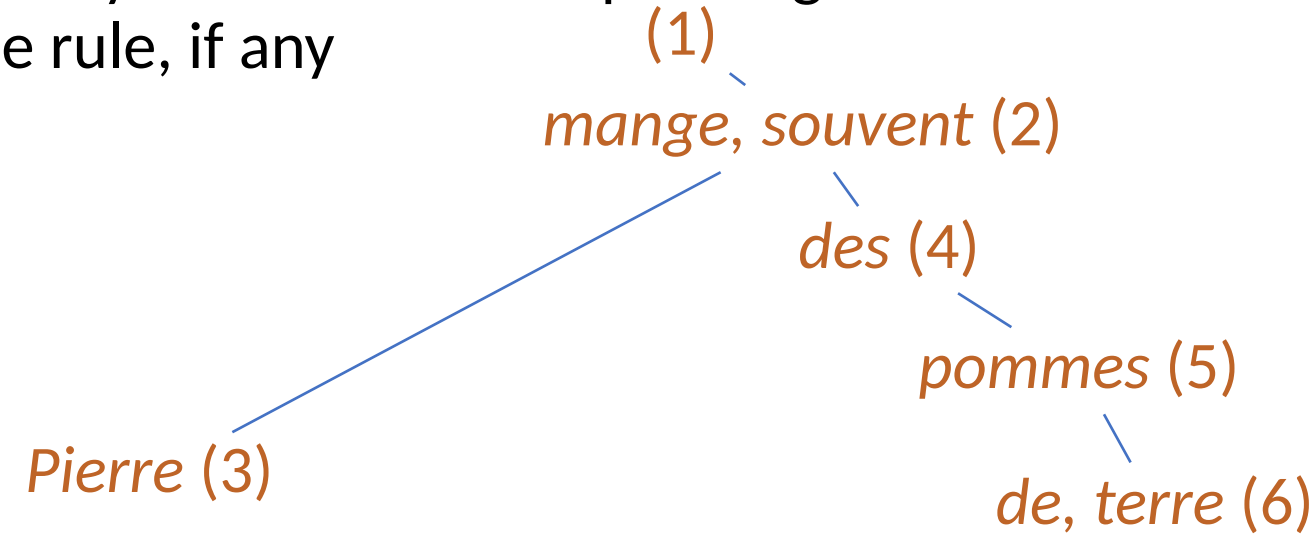
The derivation tree

- Replacing rules by the words corresponding to terminal symbols in the rule, if any



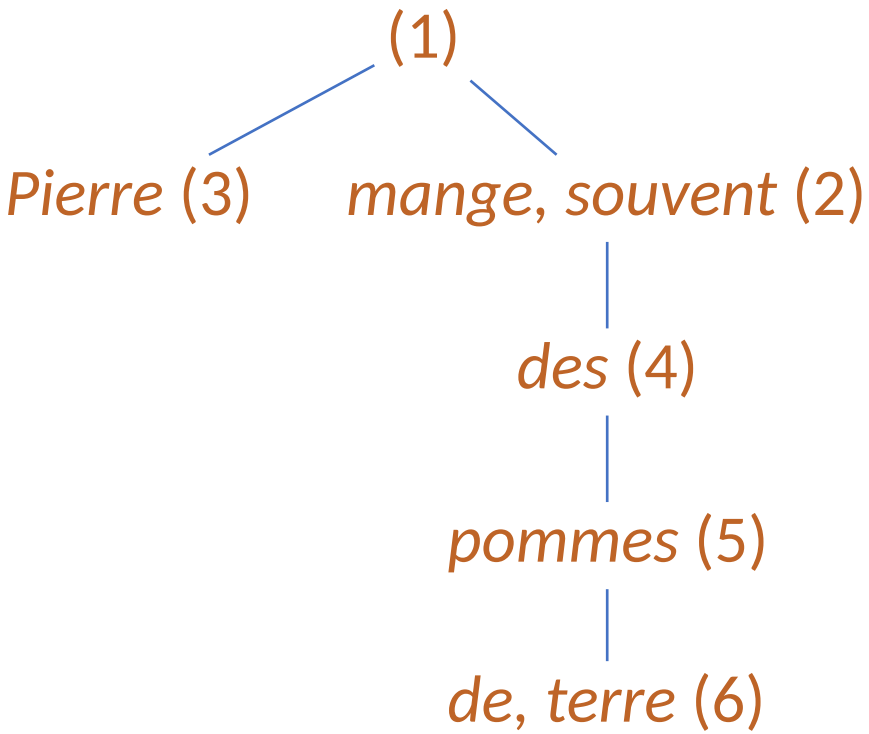
The derivation tree

- Replacing rules by the words corresponding to terminal symbols in the rule, if any



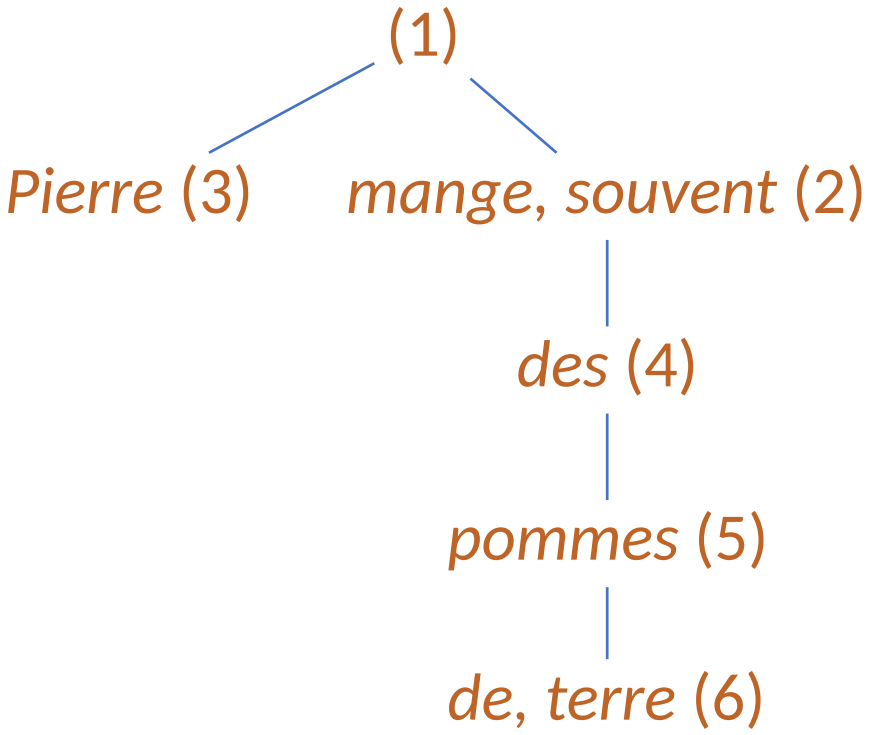
The derivation tree

- Replacing rules by the words corresponding to terminal symbols in the rule, if any



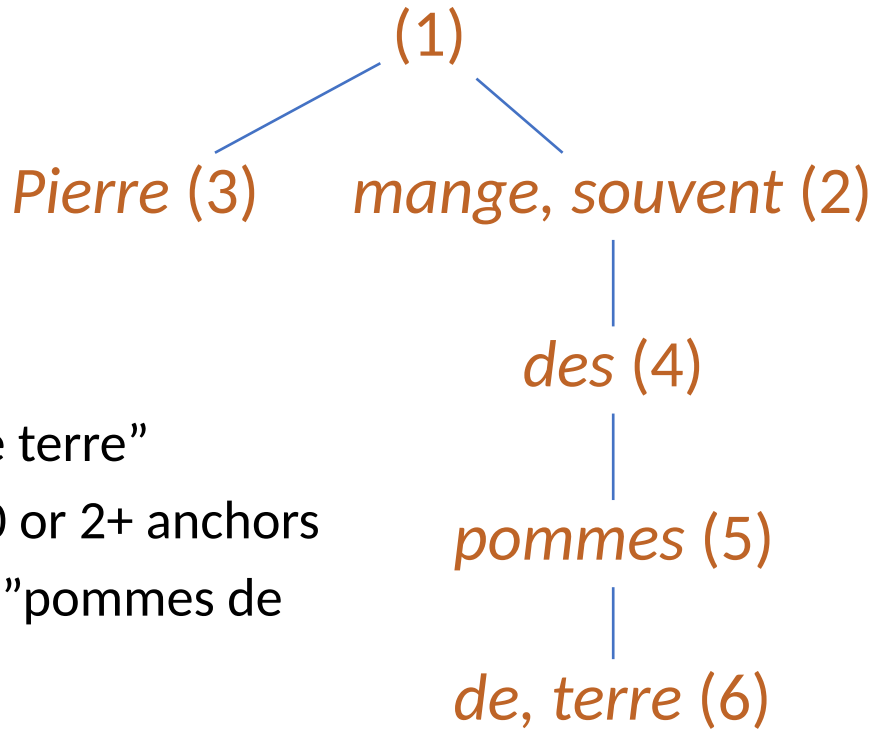
The derivation tree

- The derived tree provides us with a second structure
- What would it require for this structure to be closer to a semantic structure (~ a dependency tree)?



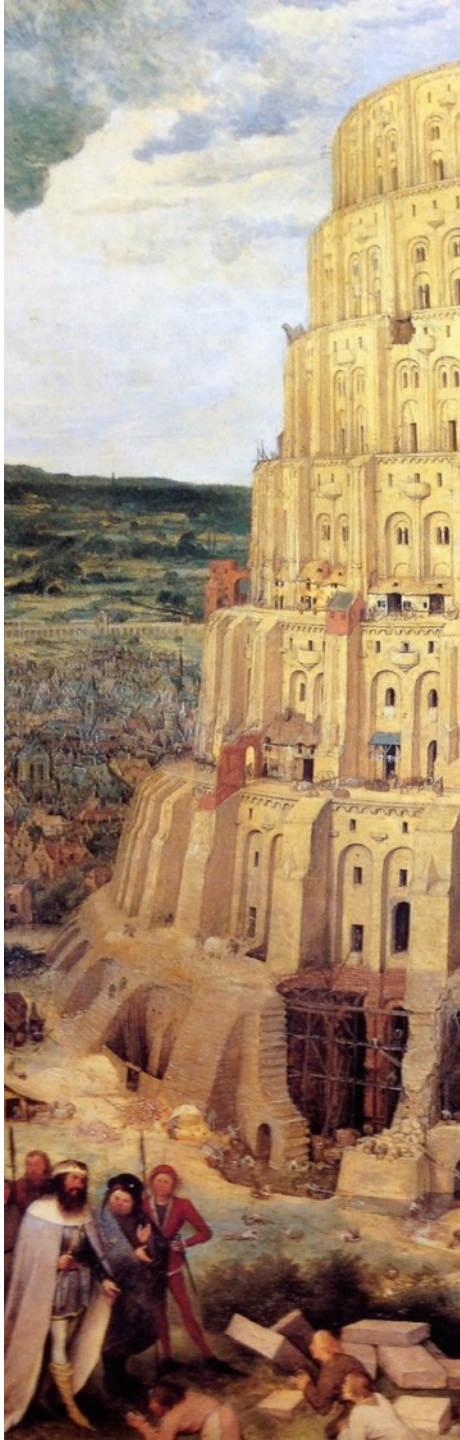
The derivation tree

- The derived tree provides us with a second structure
- What would it require for this structure to be closer to a semantic structure (~ a dependency tree)?



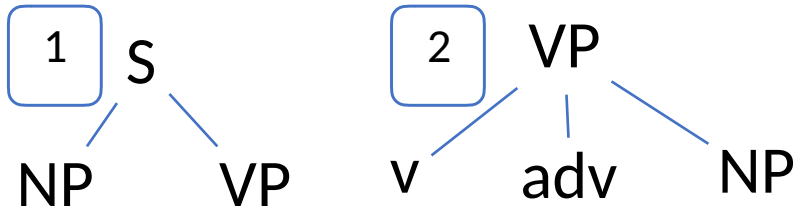
- Problems:
  - “pommes de terre”
  - nodes with 0 or 2+ anchors
  - “des” above ”pommes de terre”

Tree Substitution Grammars



Context-free grammars (CFGs)

- We replace rewriting rules by equivalent "**elementary trees**"



1 S ⟶ NP VP

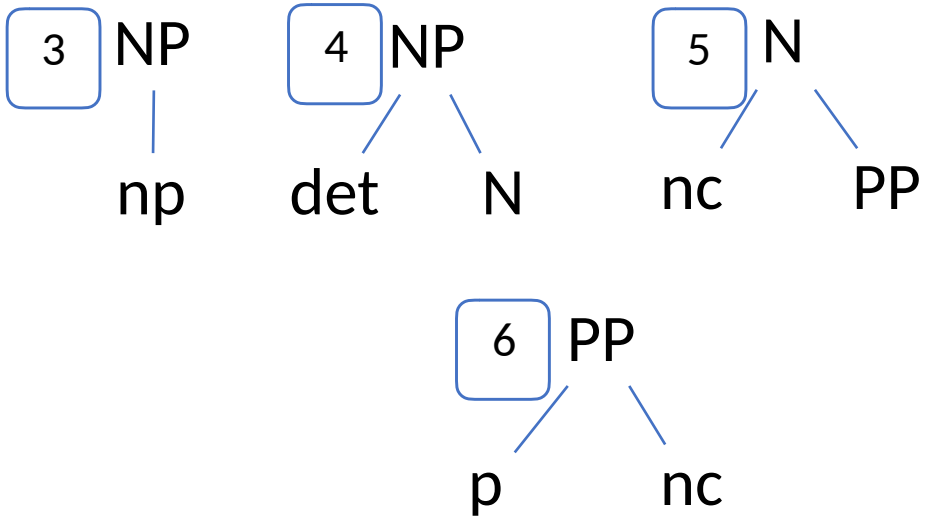
2 VP ⟶ v adv NP

3 NP ⟶ np

4 NP ⟶ det N

5 N ⟶ nc PP

6 PP ⟶ p nc



Context-free grammars (CFGs)

- We replace rewriting rules by equivalent "**elementary trees**"

1 S ⟶ NP VP

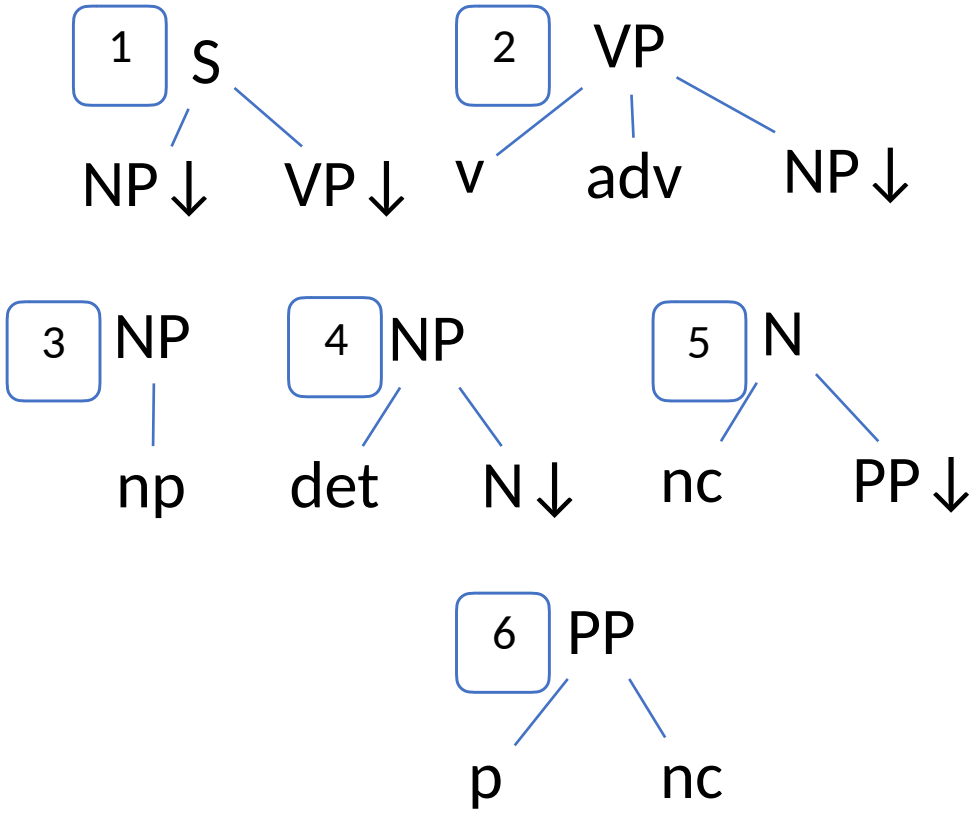
2 VP ⟶ v adv NP

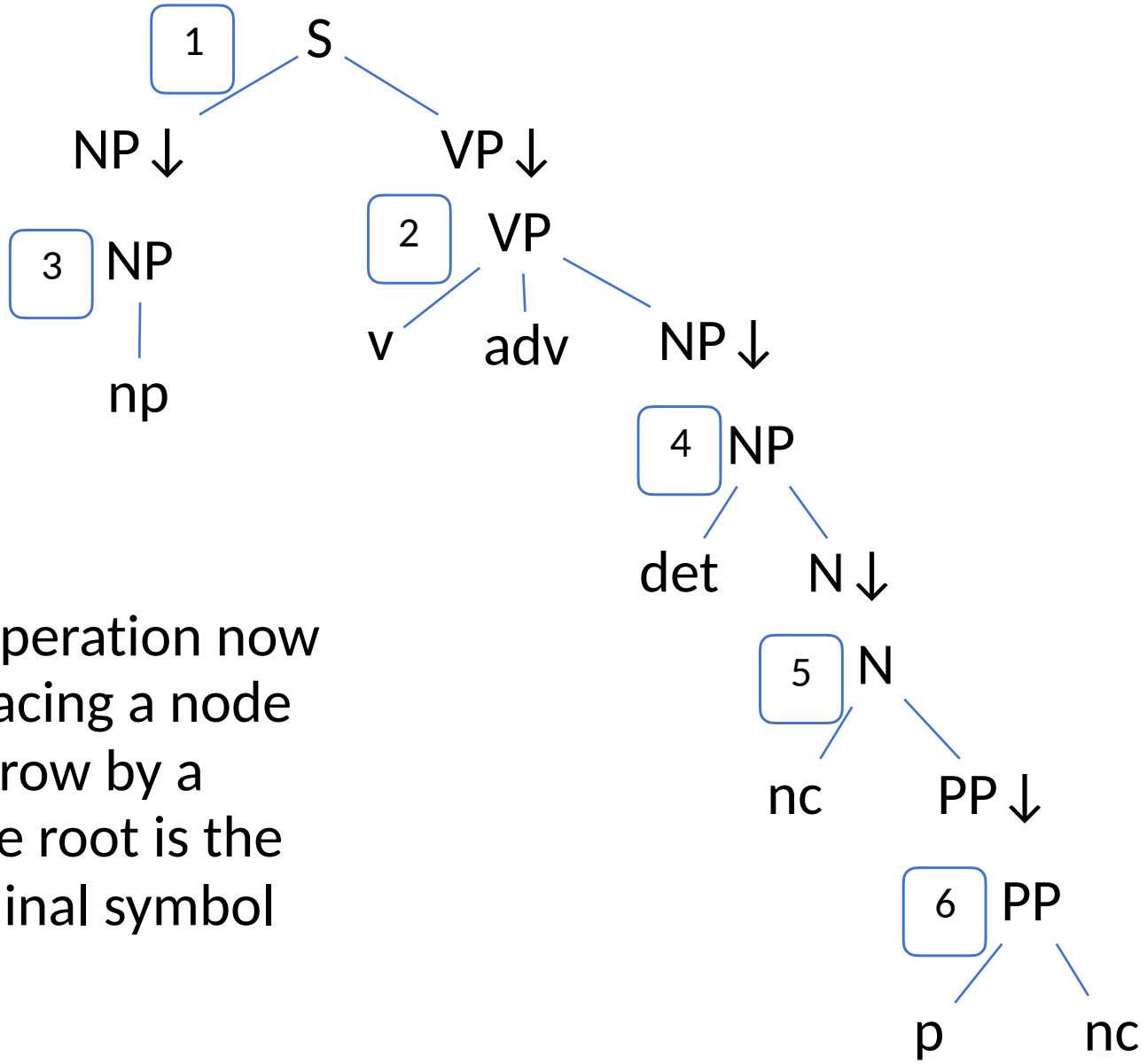
3 NP ⟶ np

4 NP ⟶ det N

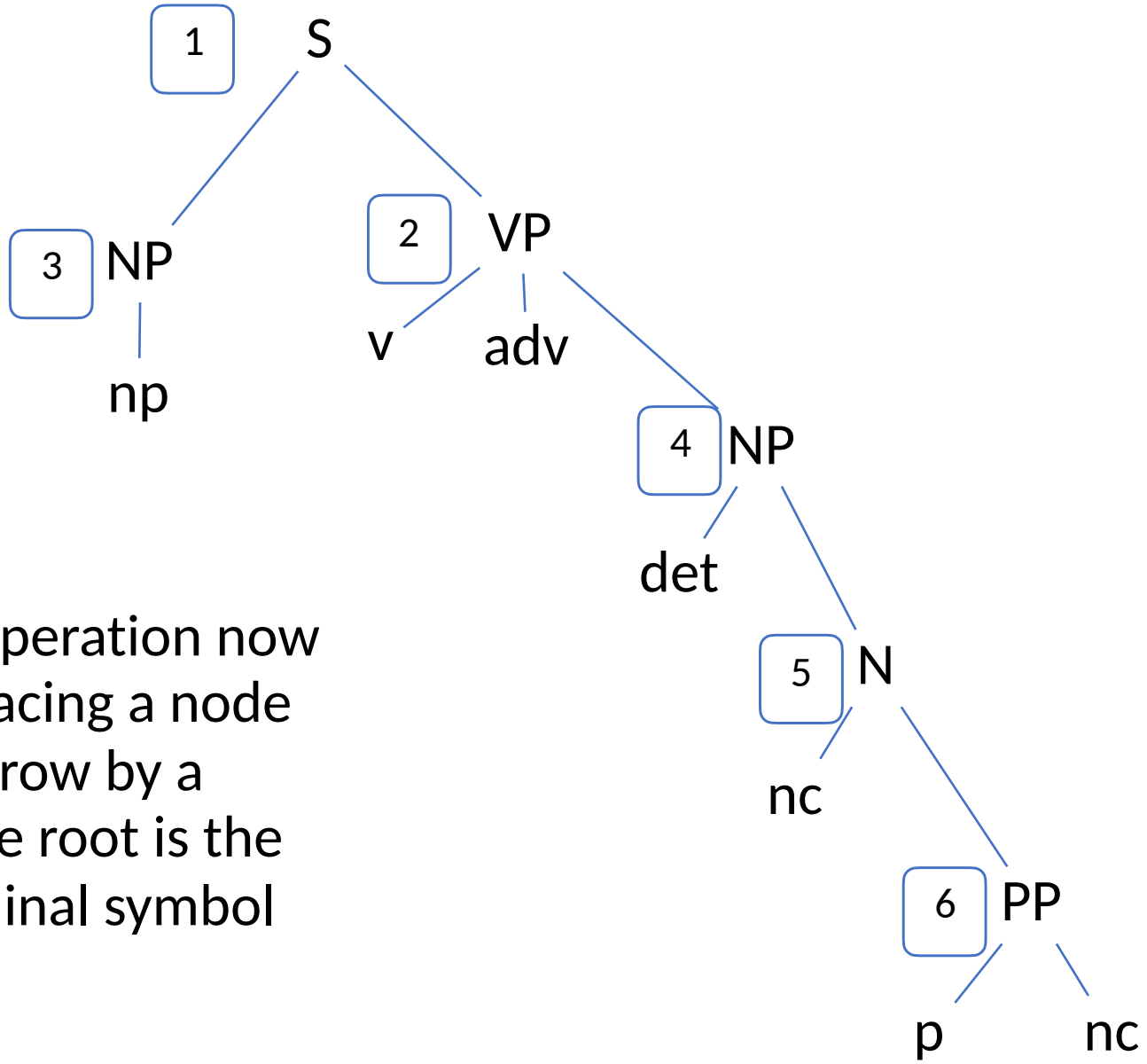
5 N ⟶ nc PP

6 PP ⟶ p nc

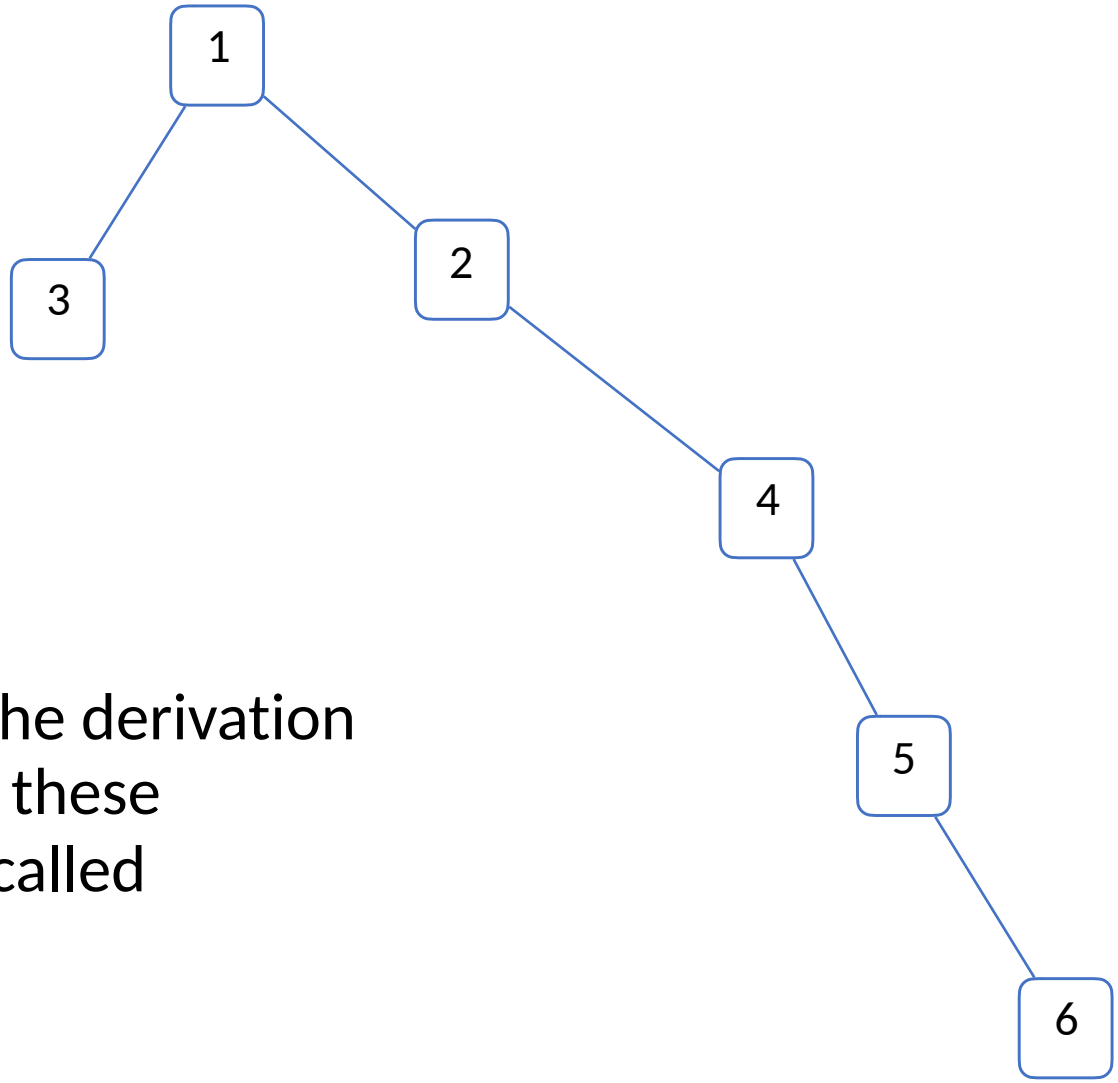




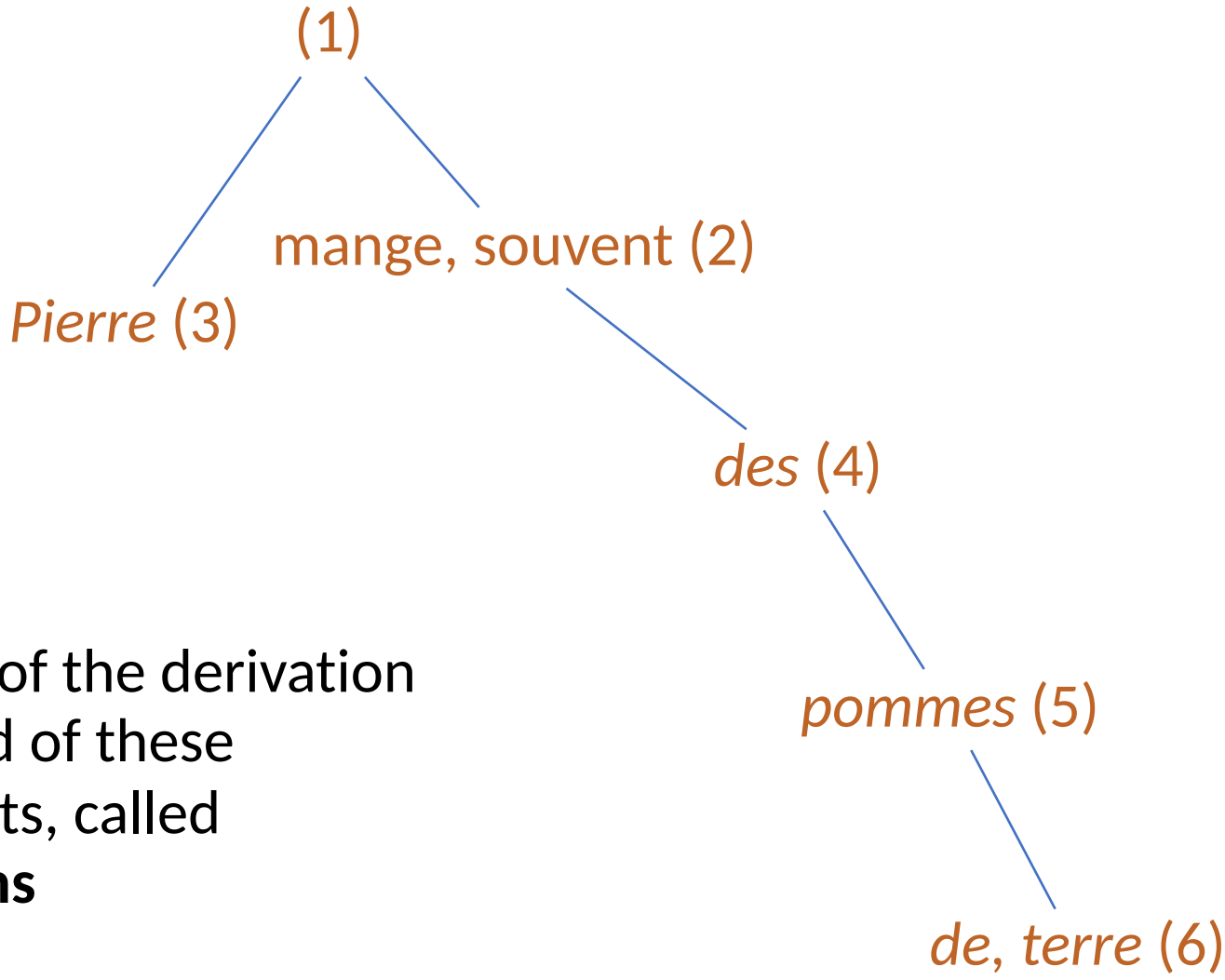
- The rewriting operation now consists in replacing a node with a down-arrow by a (sub)tree whose root is the same non-terminal symbol



- The rewriting operation now consists in replacing a node with a down-arrow by a (sub)tree whose root is the same non-terminal symbol



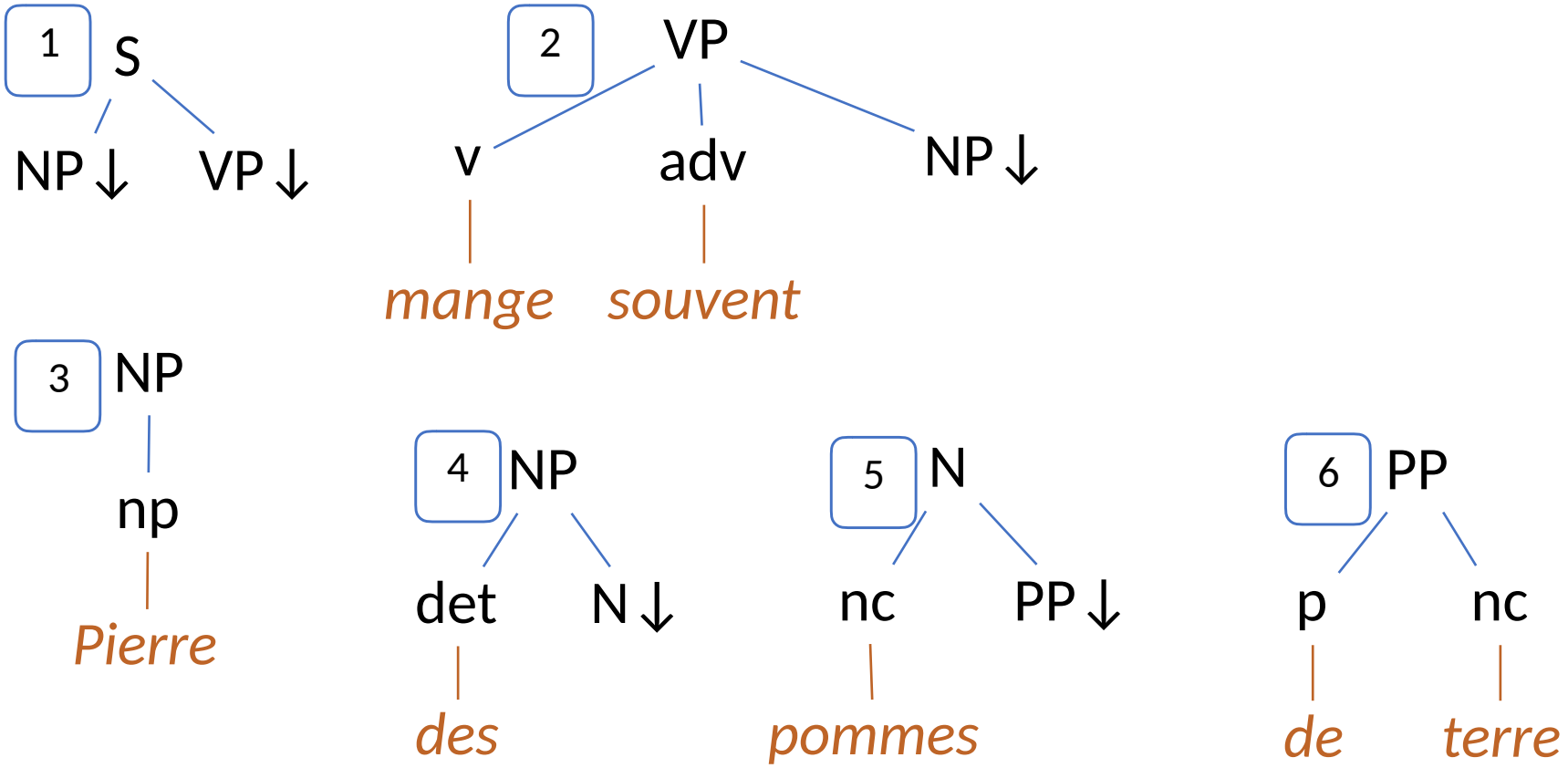
- The history of the derivation is the record of these replacements, called **substitutions**



- The history of the derivation is the record of these replacements, called **substitutions**

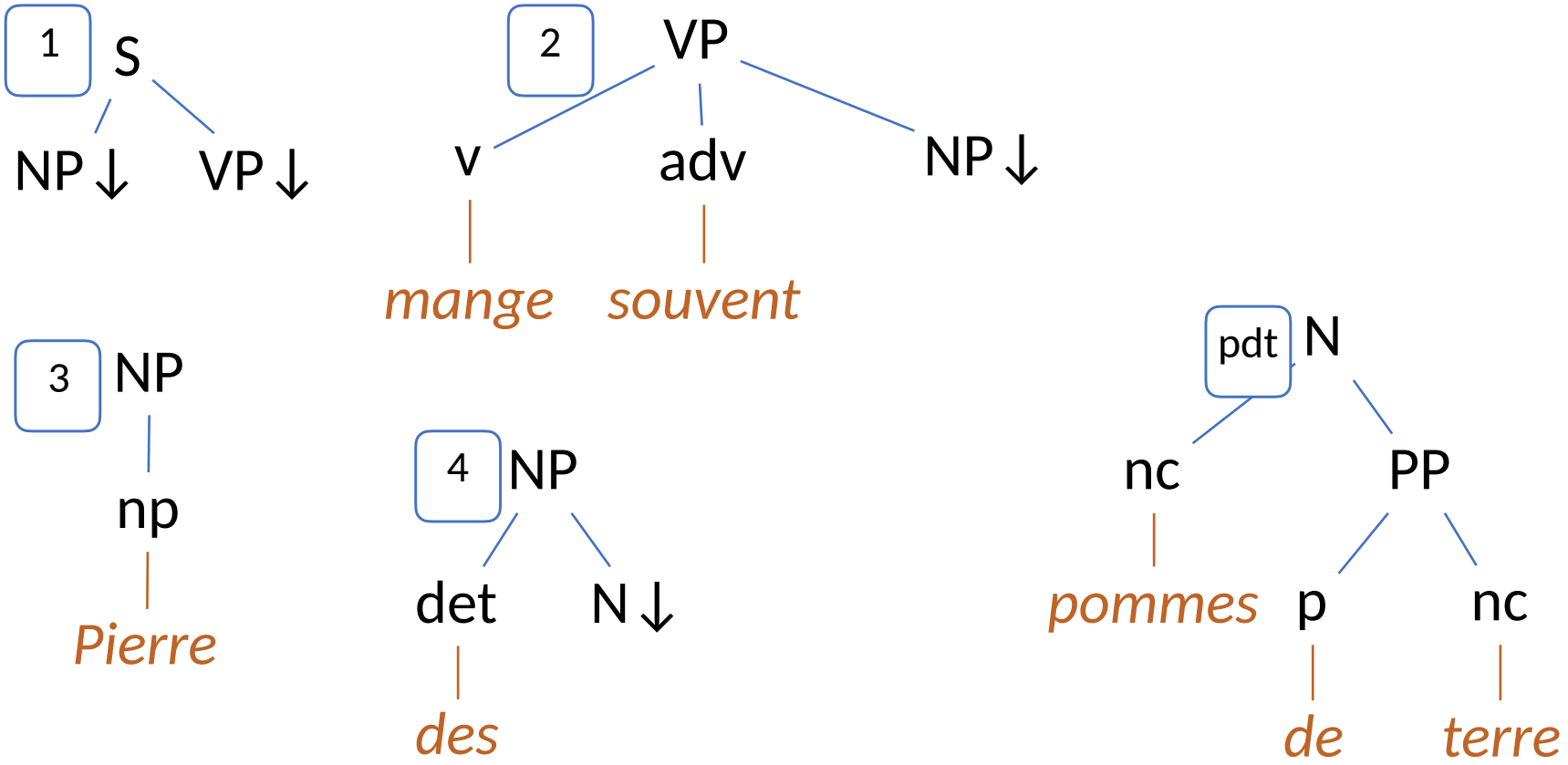
Tree substitution grammars

- Let us reintroduce the lexicon (i.e. anchoring)

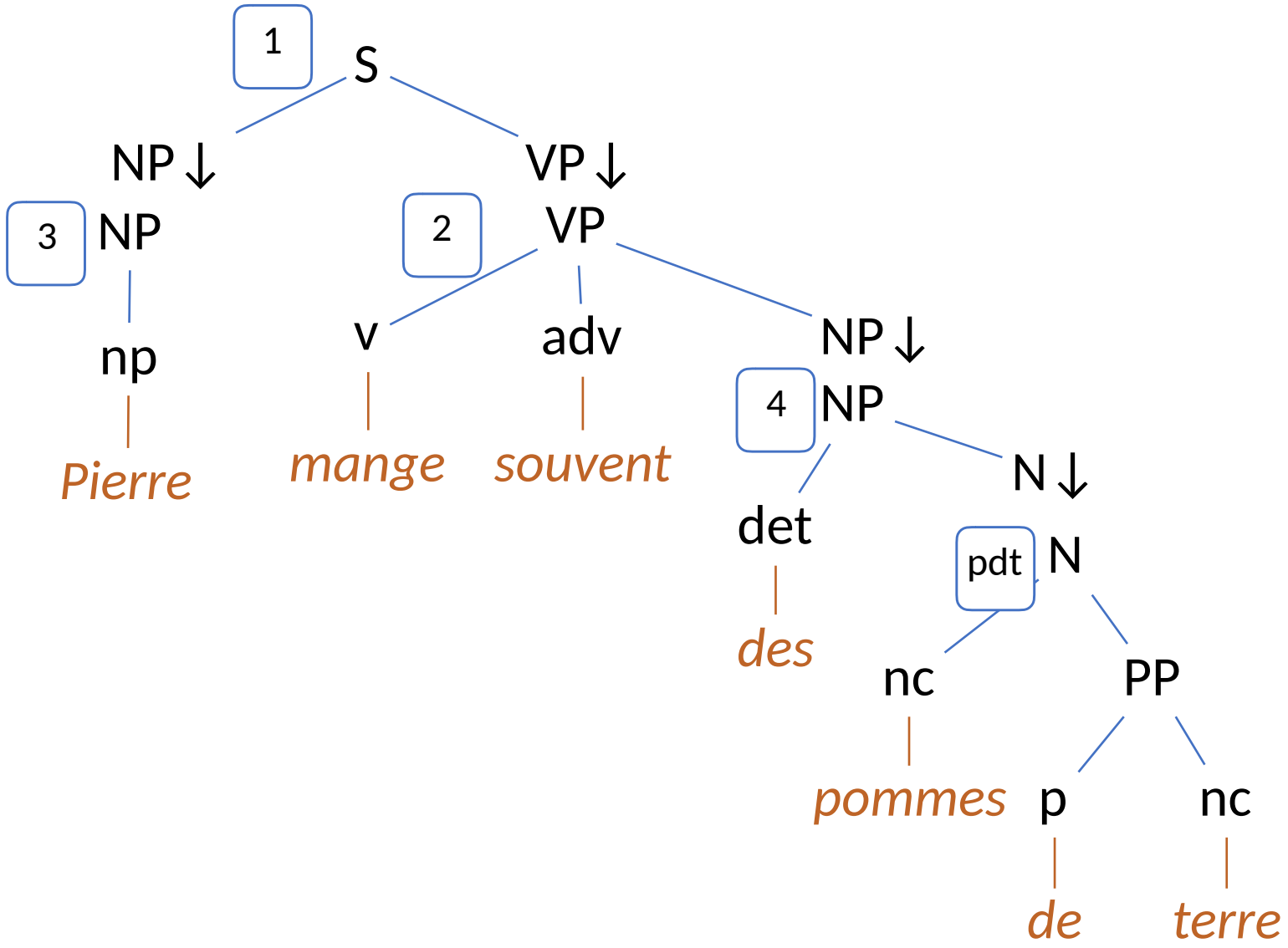


Tree substitution grammars

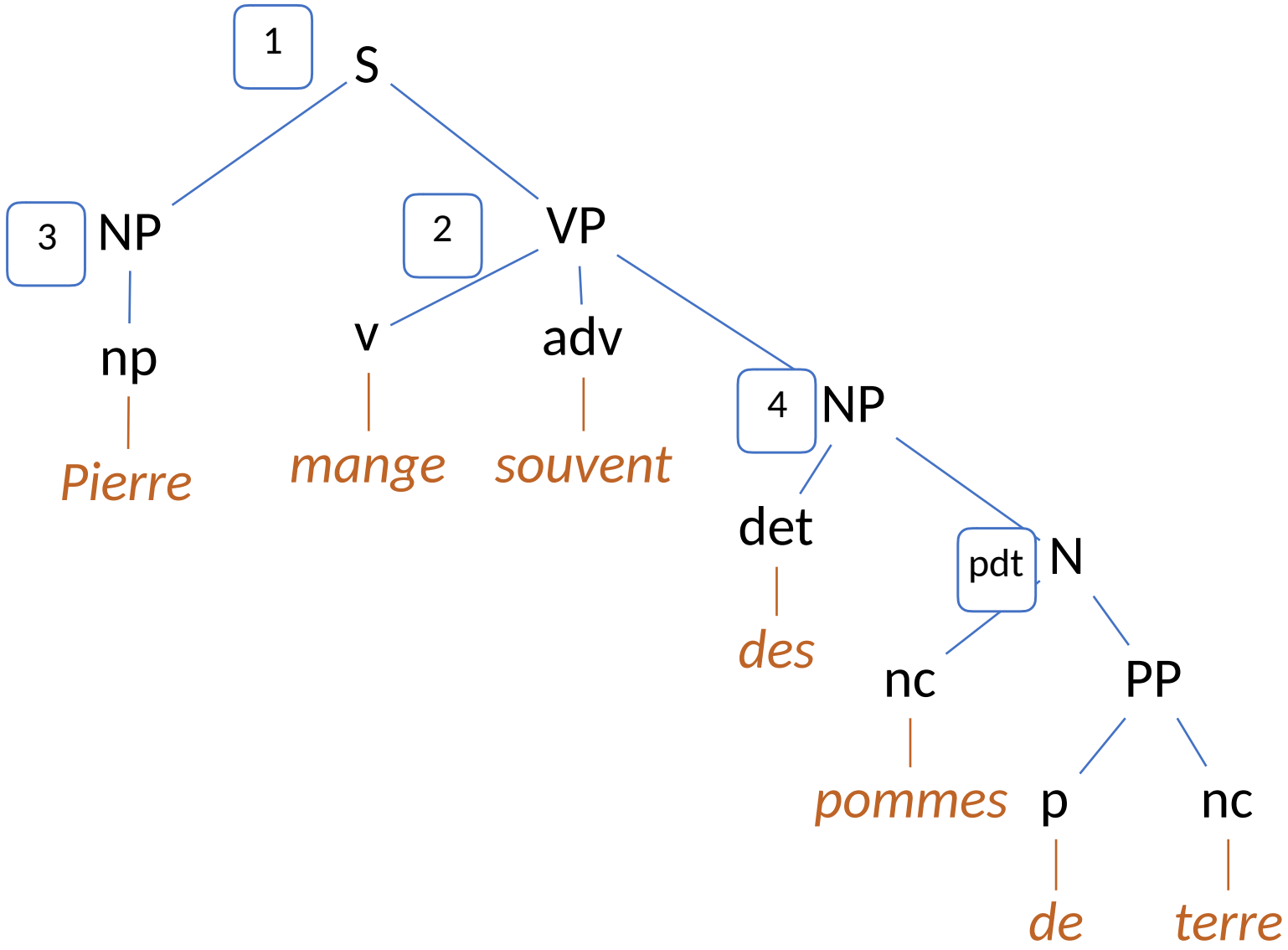
- There is no reason to stick to 2-level elementary trees



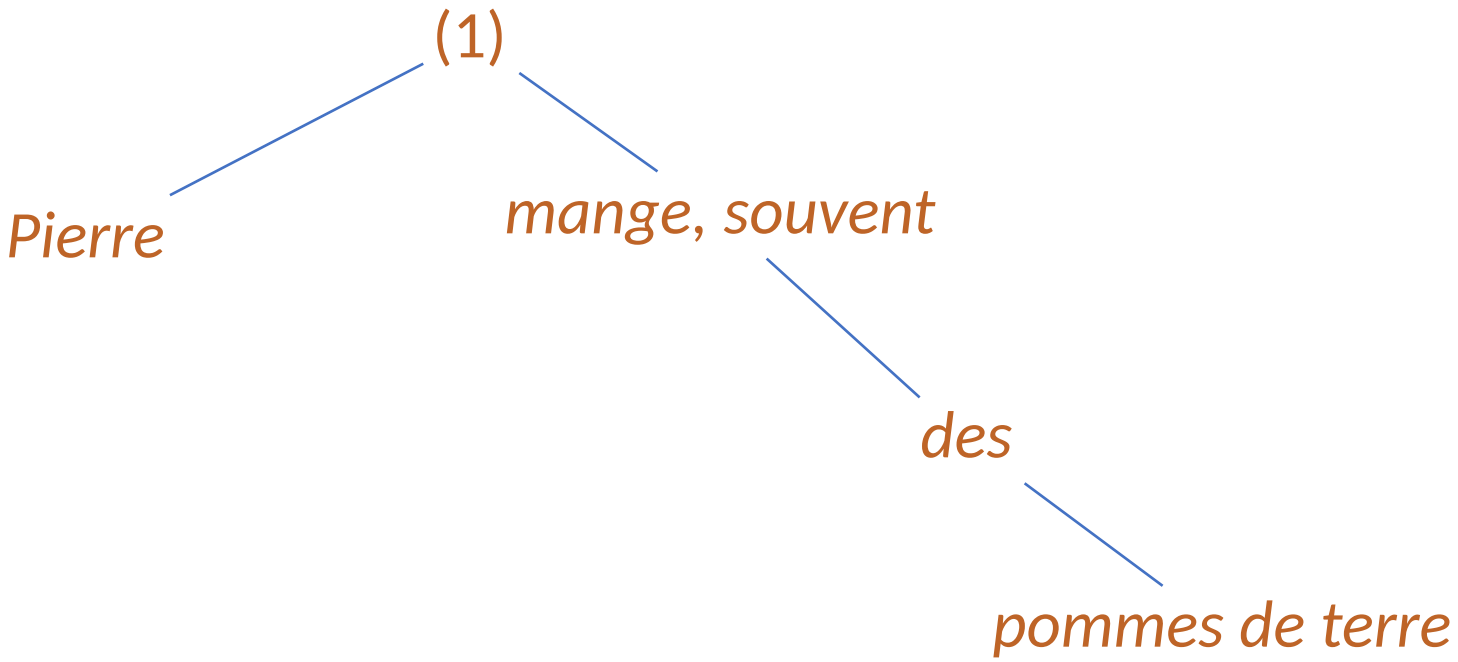
Solving the compound word problem



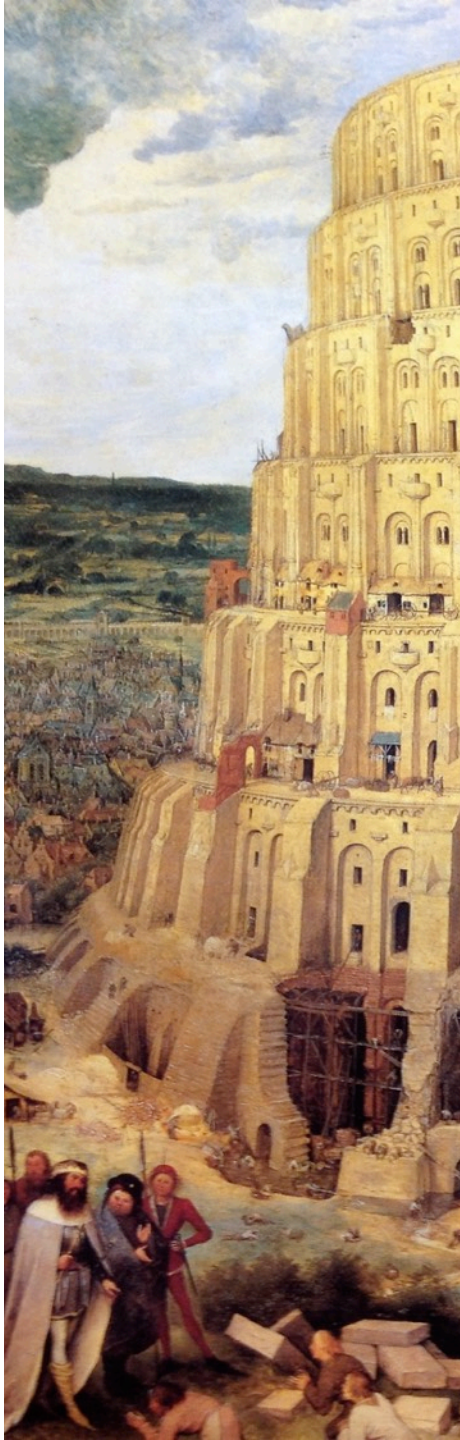
Solving the compound word problem



Solving the compound word problem



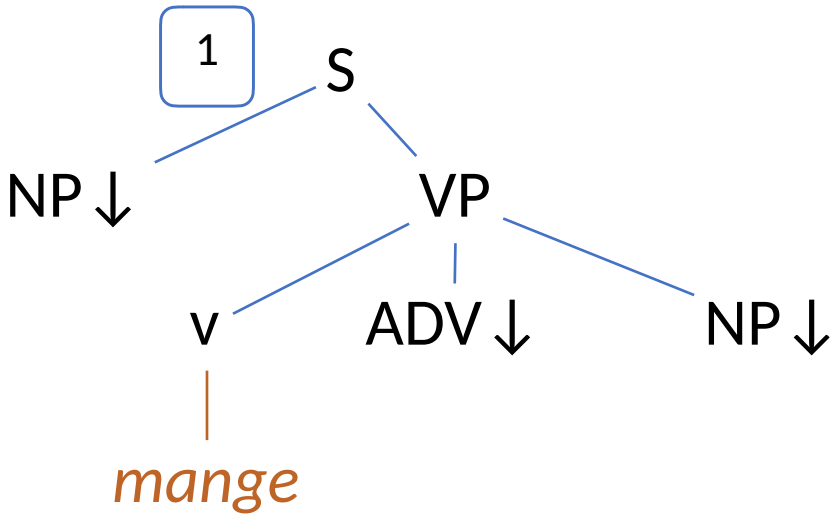
Lexicalisation

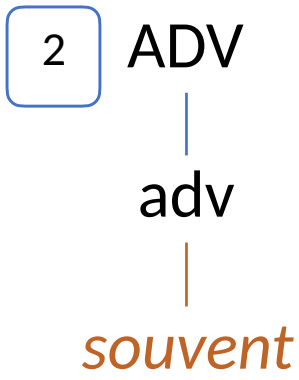


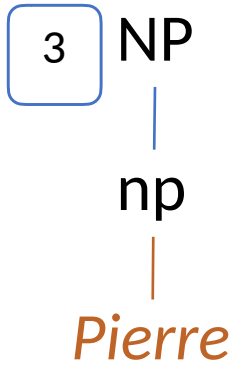
Lexicalisation

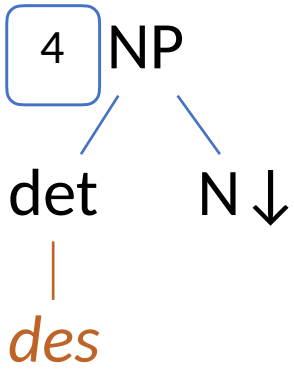
- In order to avoid non-anchored nodes, we can ensure that every elementary tree has **at least one terminal symbol**
- Moreover, we can ensure that every tree has **anchors that constitute a semantic word**
- We will once again take advantage of the fact that TSGs do not bound the depth of elementary trees (contrary to CFGs)

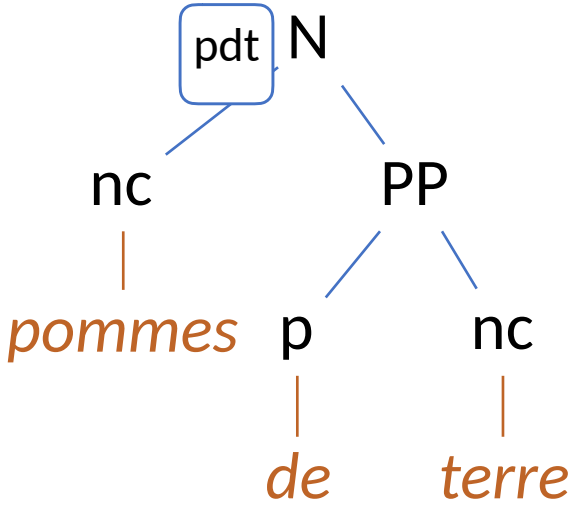
Lexicalised tree substitution grammar



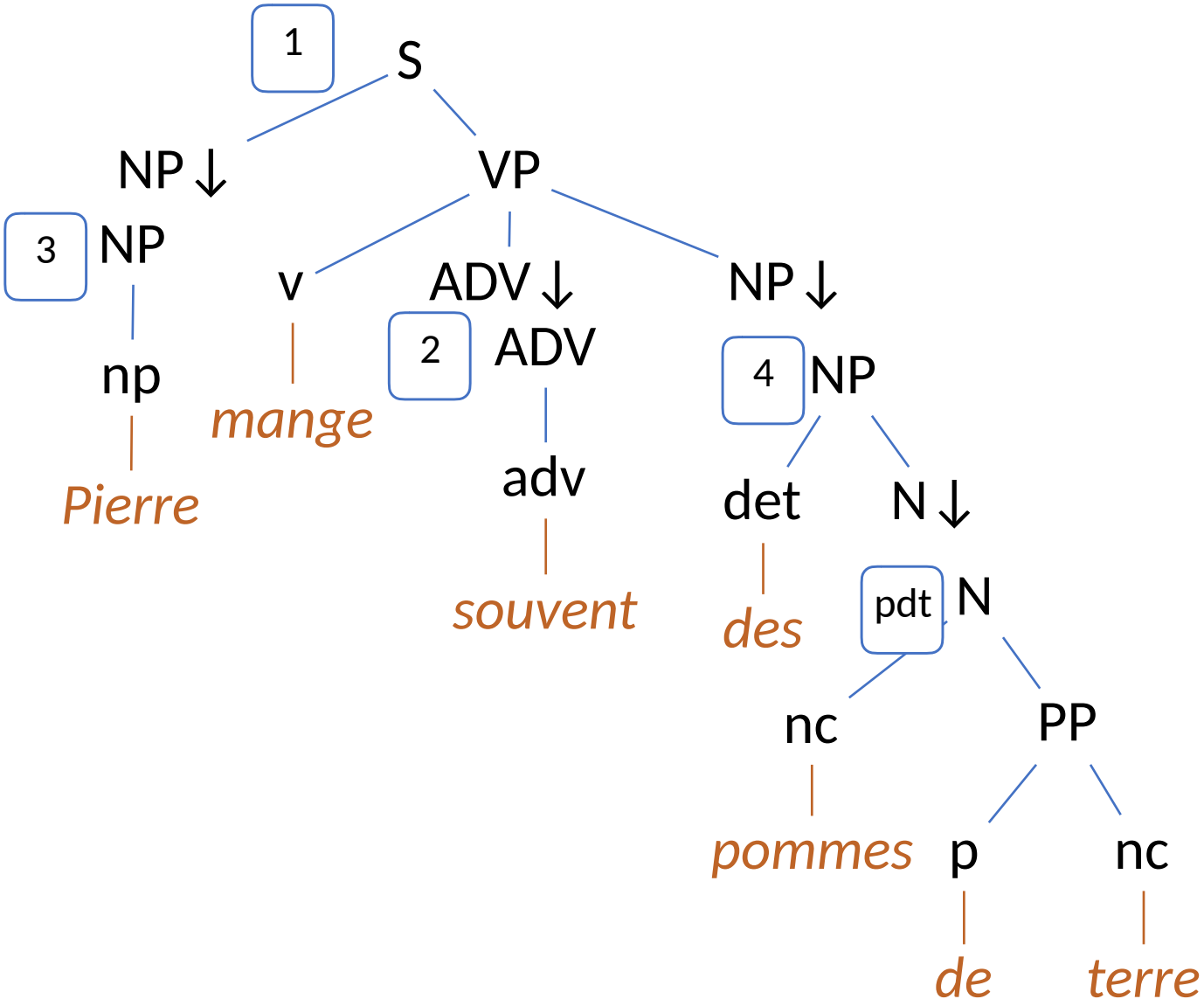




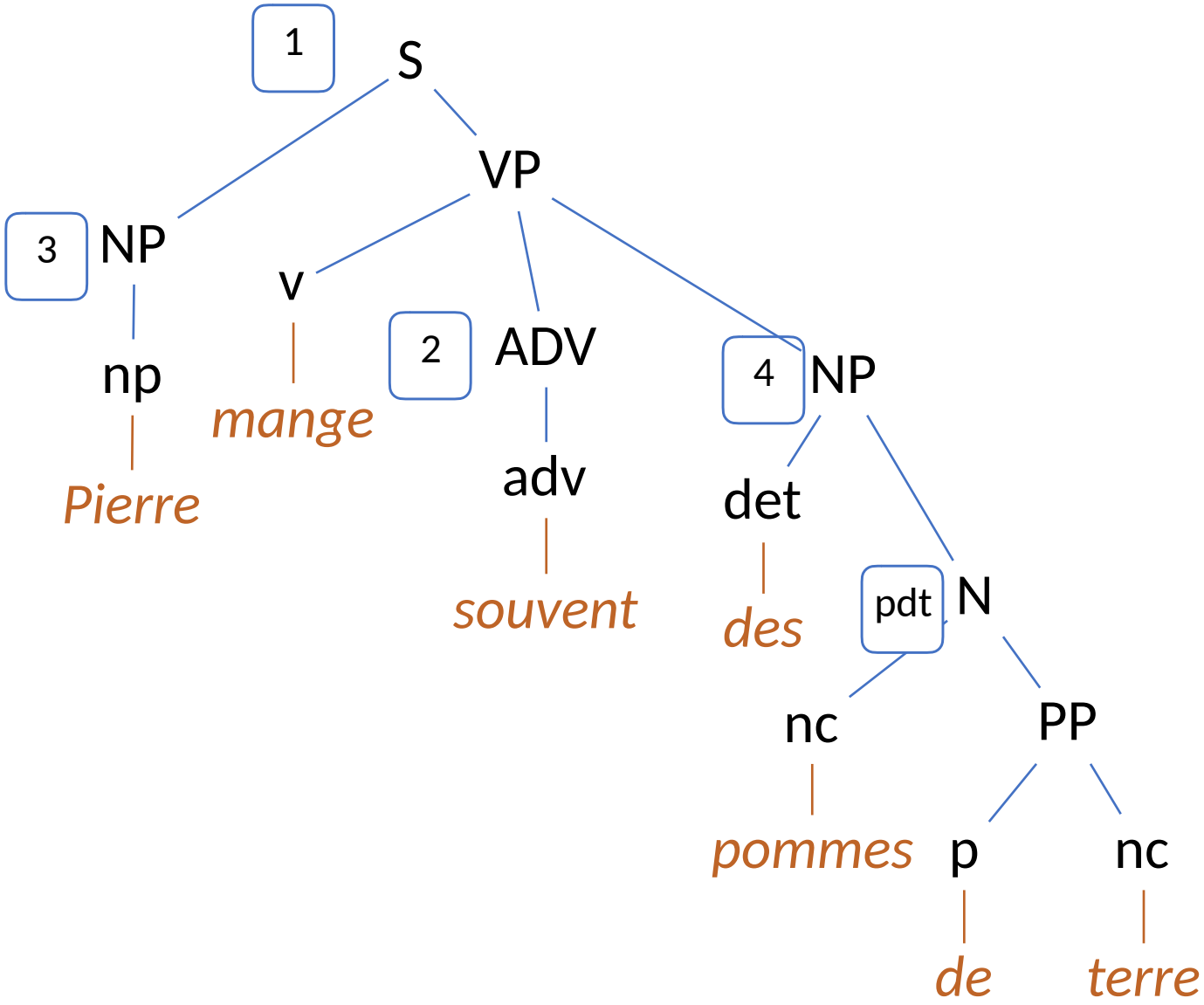




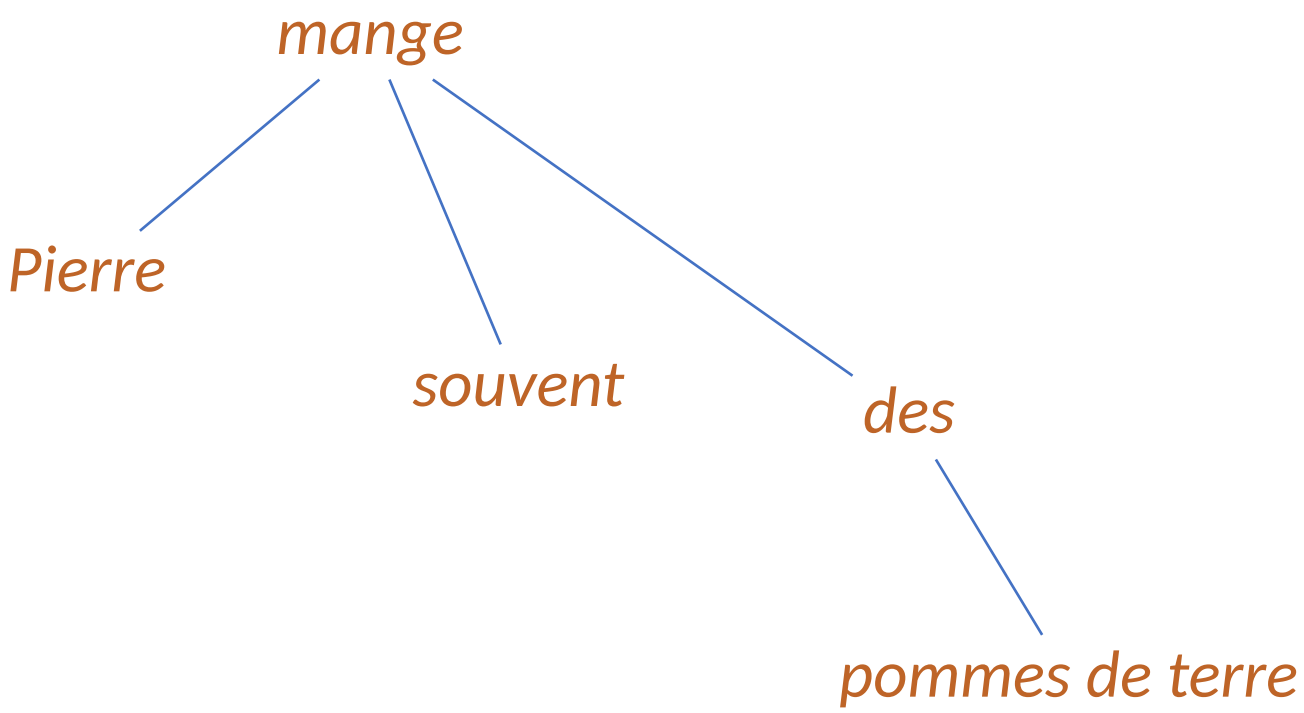
Lexicalised tree substitution grammar



Lexicalised tree substitution grammar



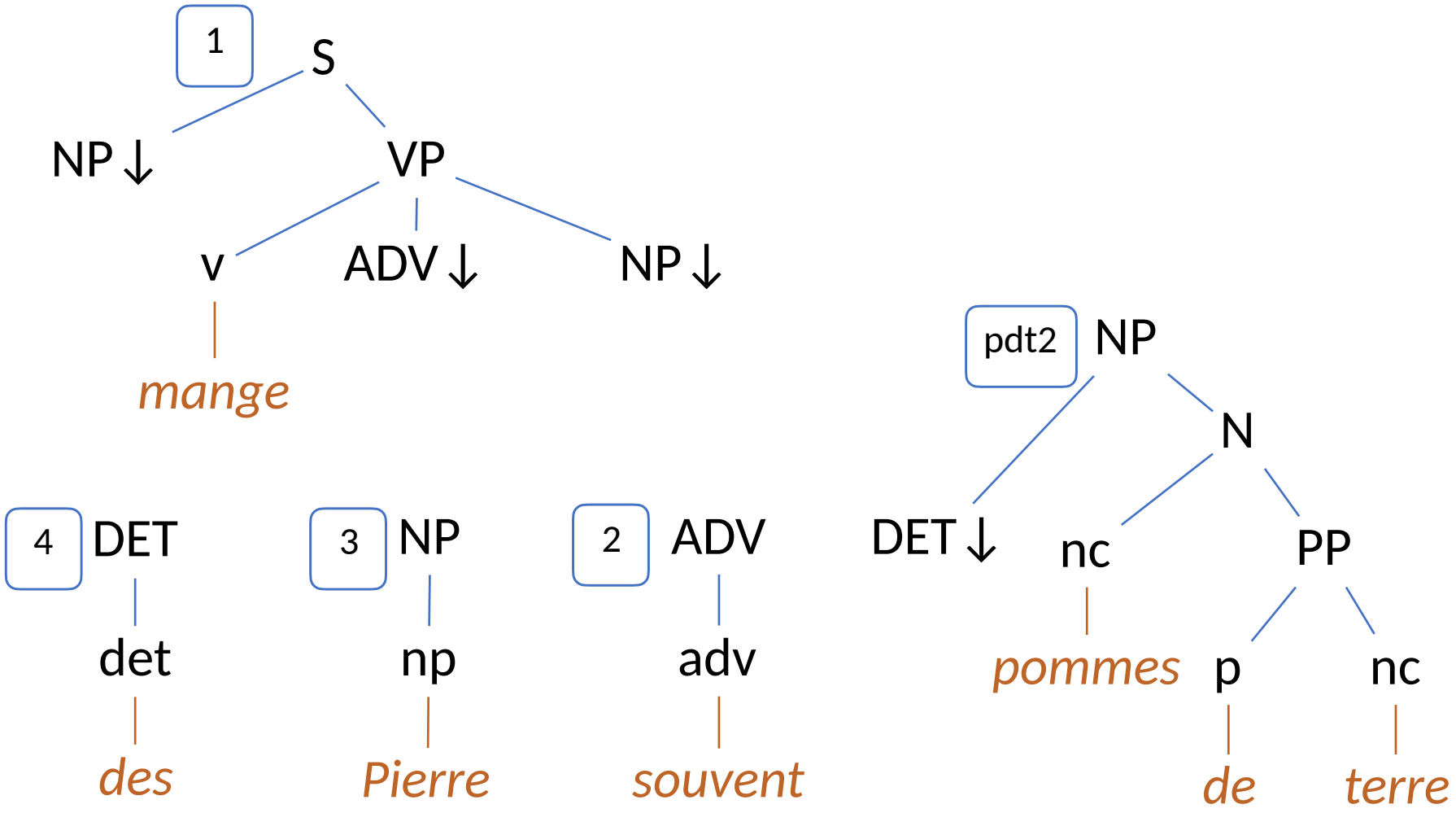
Lexicalised tree substitution grammar



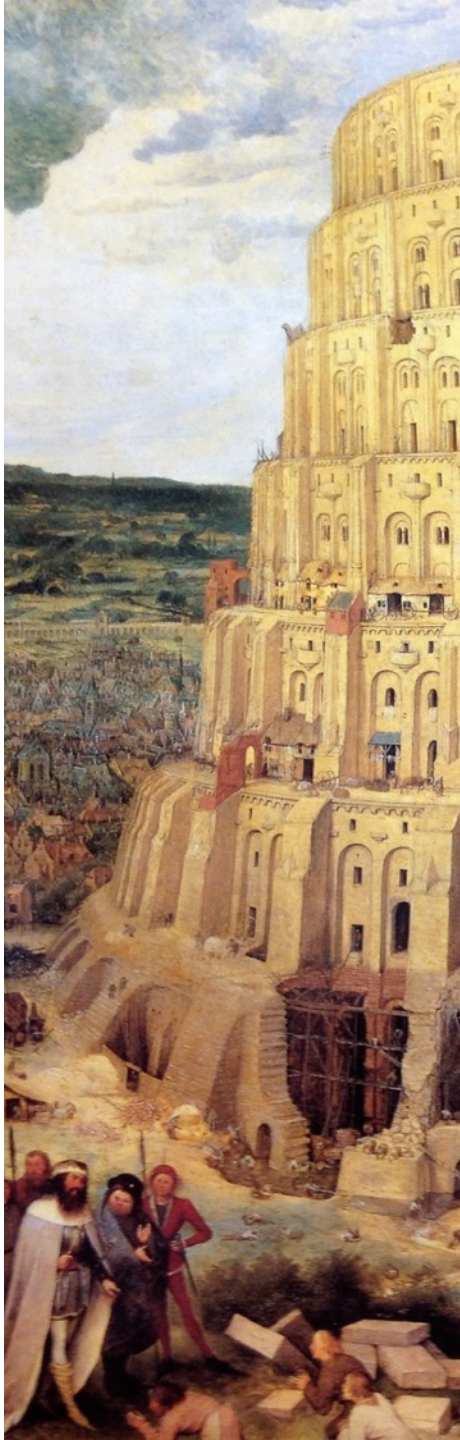
Lexicalisation

- Slight improvement: a new non-terminal to get the determiner *below* its noun

Lexicalisation



Tree adjunction grammars



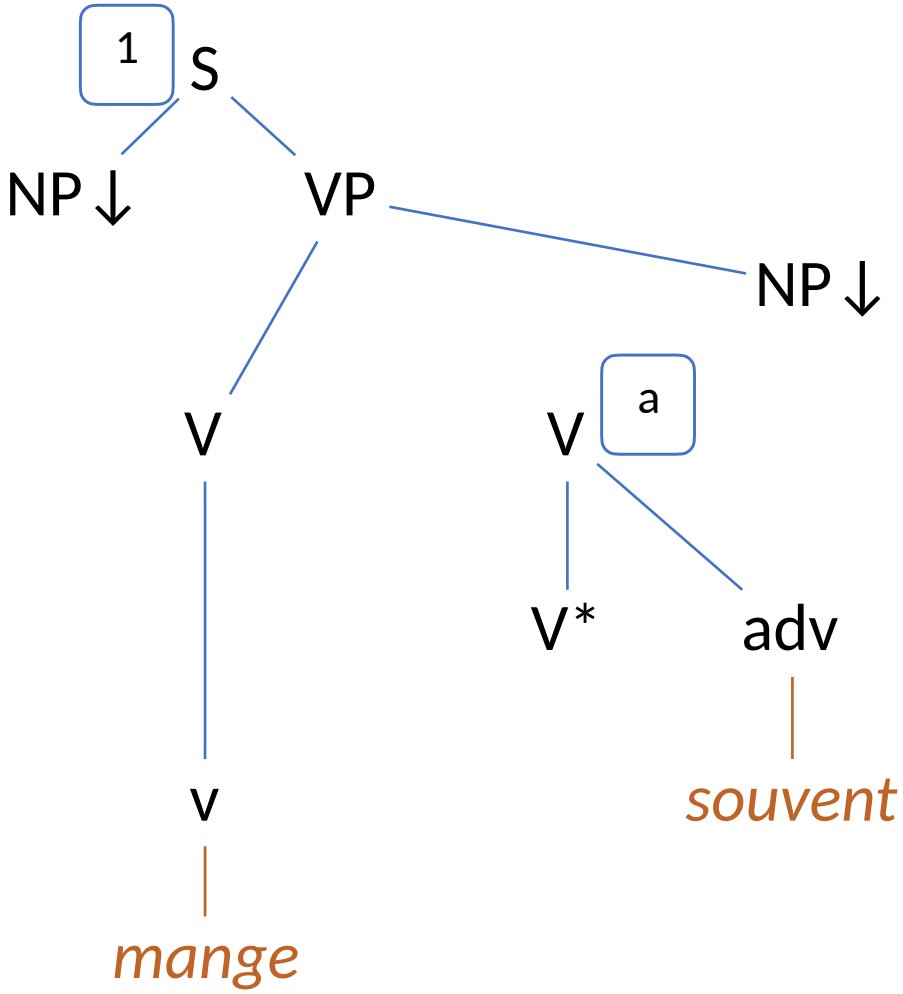
Arguments and modifiers

- In syntax, two major categories of dependencies are distinguished: **arguments and modifiers**
  - The presence or absence and the properties of an argument are determined by its governor
  - A modifier is a self-standing phrase that modifies its governor
- Generally, a direct object is an argument, whereas a circumstantial complement (e.g. expressing a location) is a modifier
  - In our example, *souvent* ‘often’ is a modifier of *mange* ‘eat’, whereas *Pierre* and *des pommes de terre* ‘potatoes’ are arguments of *mange*

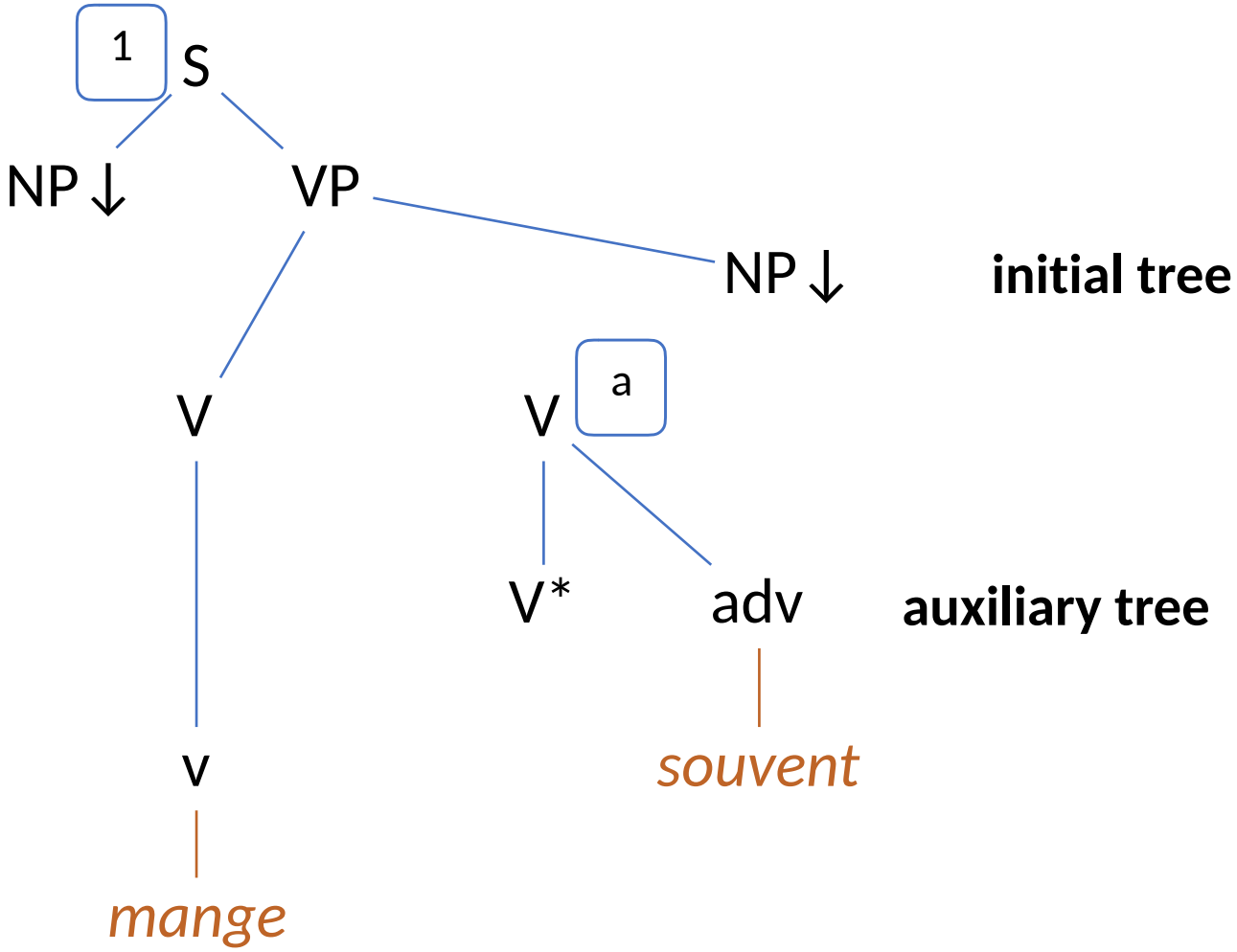
Arguments and modifiers

- Ideally, we would need a way to distinguish arguments and modifiers
  - We would like **arguments** to be “requested” in the elementary tree of the governor => **substitution nodes**
  - We would like a mechanism for modifiers to actually “attach” by themselves to their governor, which would work as a host
- It would be a more elegant way to solve our “determiner” issue
- We will define a new operation, on top of substitution, thus defining Tree Adjoining Grammars (TAGs)
  - LTAGs when they are lexicalised

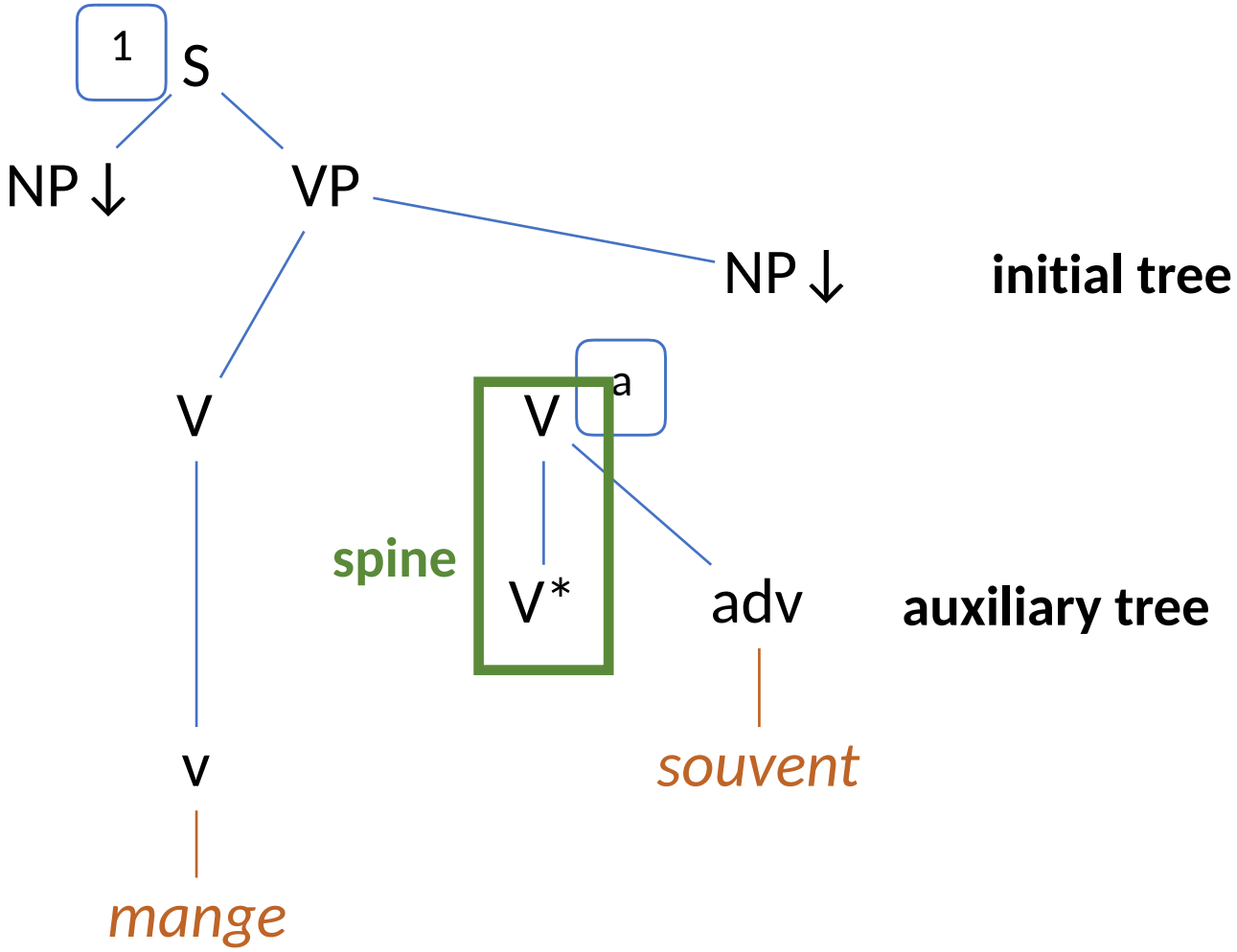
The adjunction operation



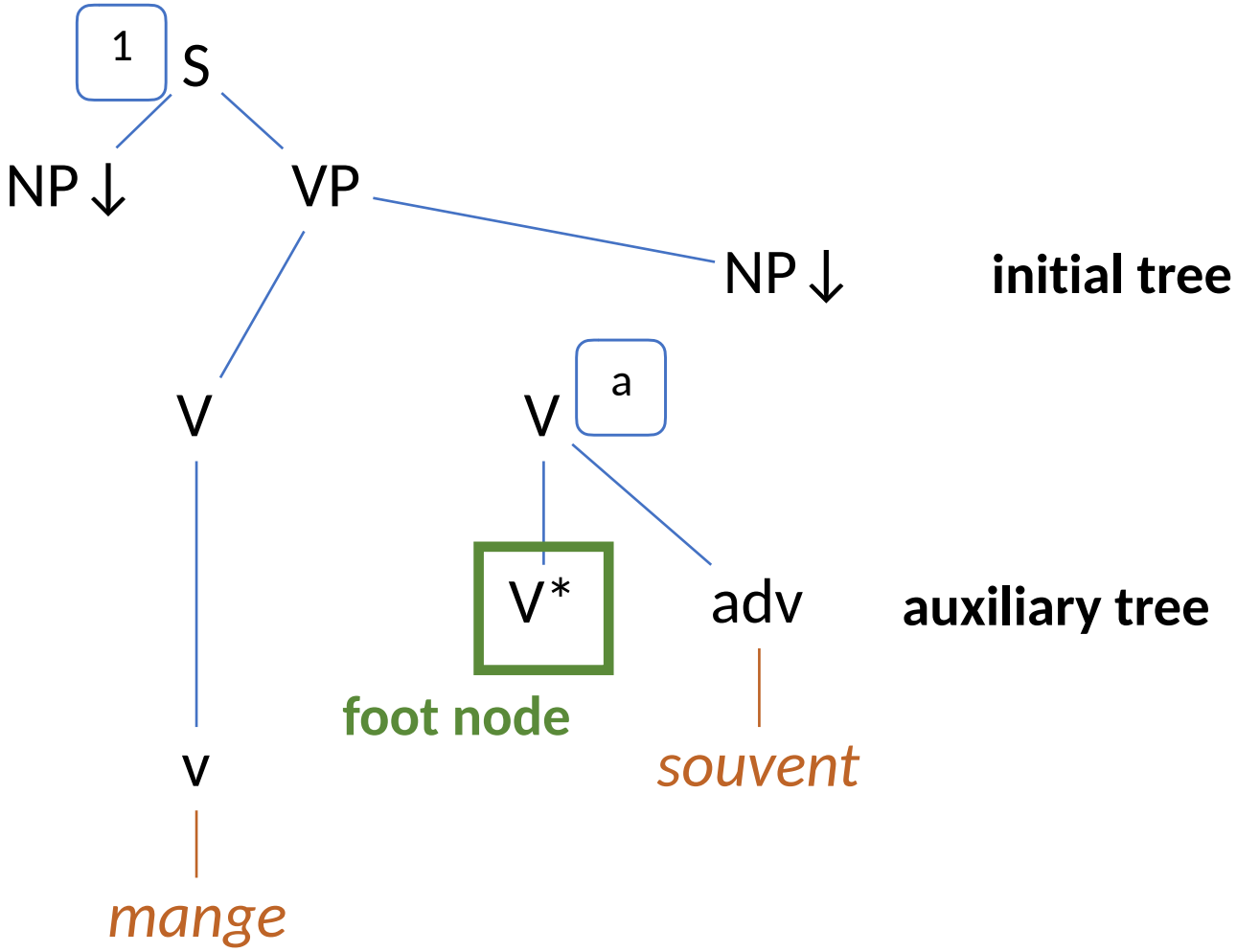
The adjunction operation



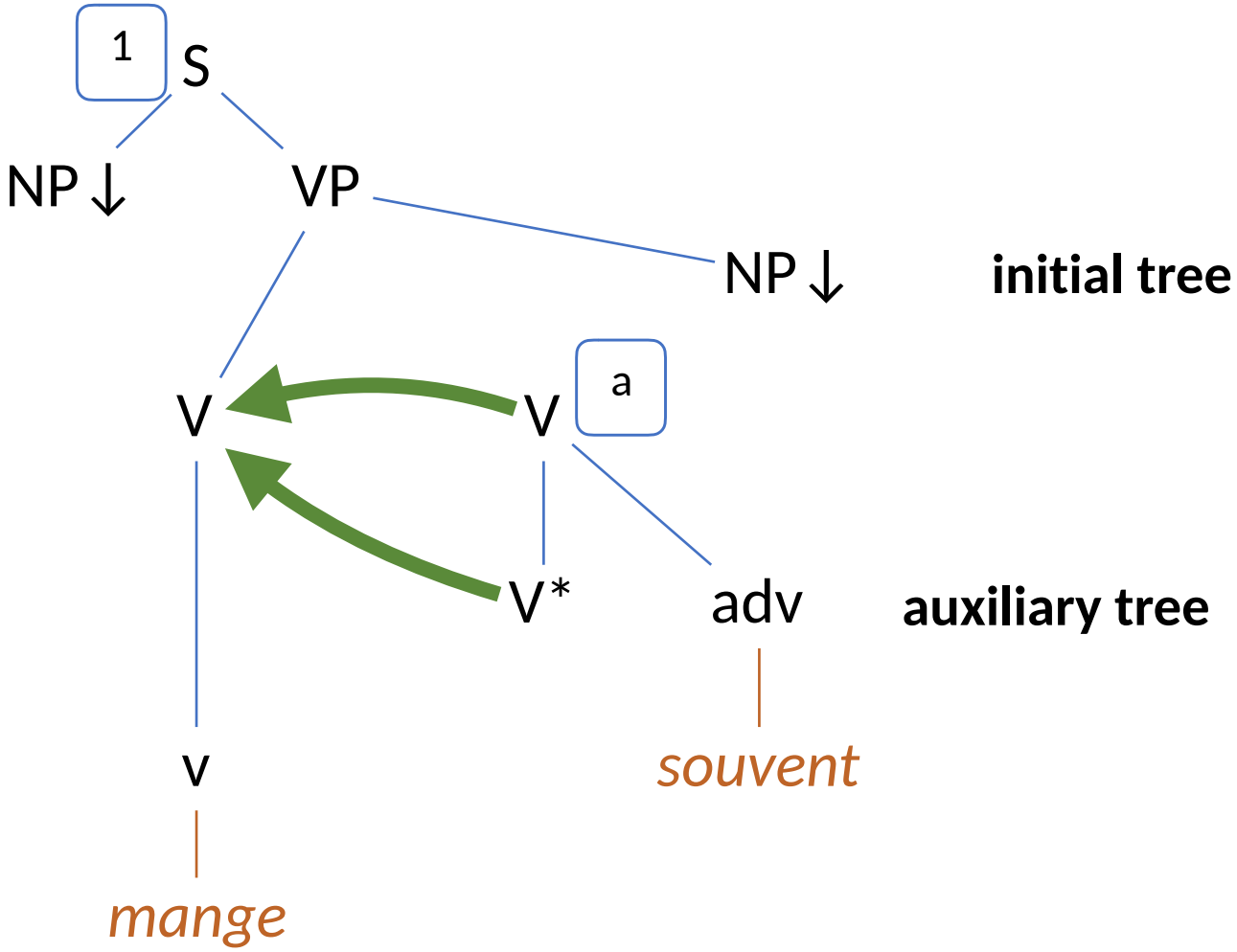
The adjunction operation



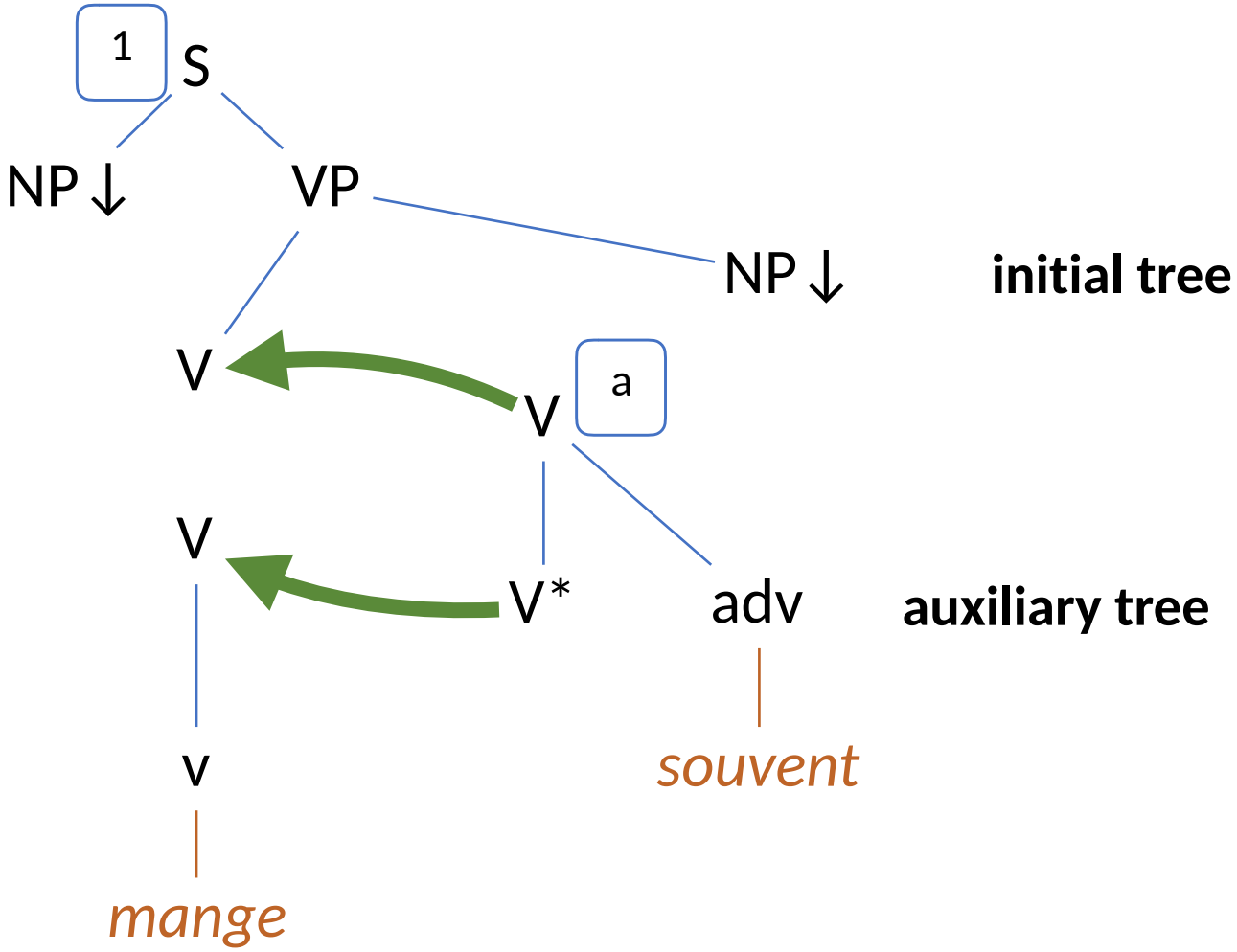
The adjunction operation



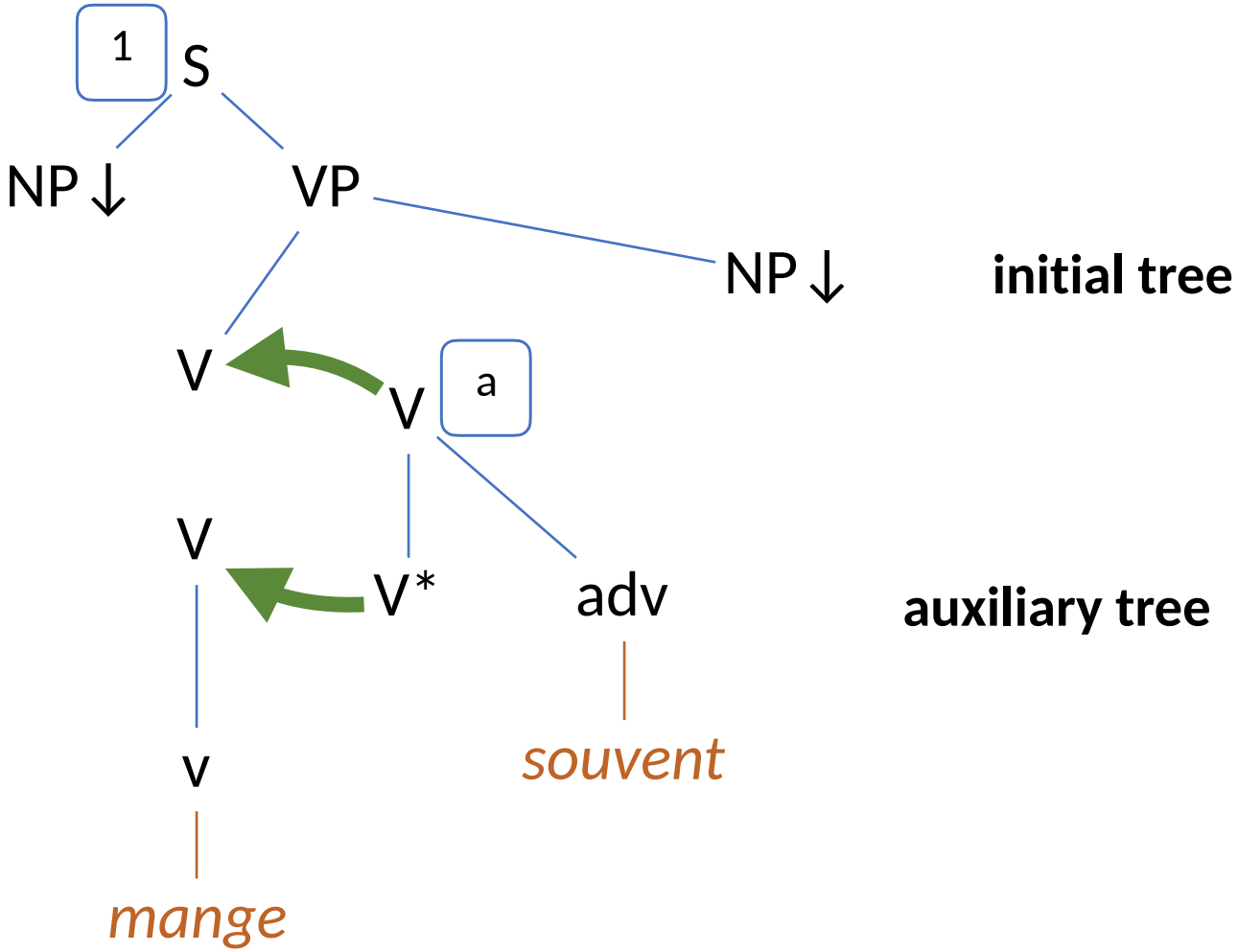
The adjunction operation



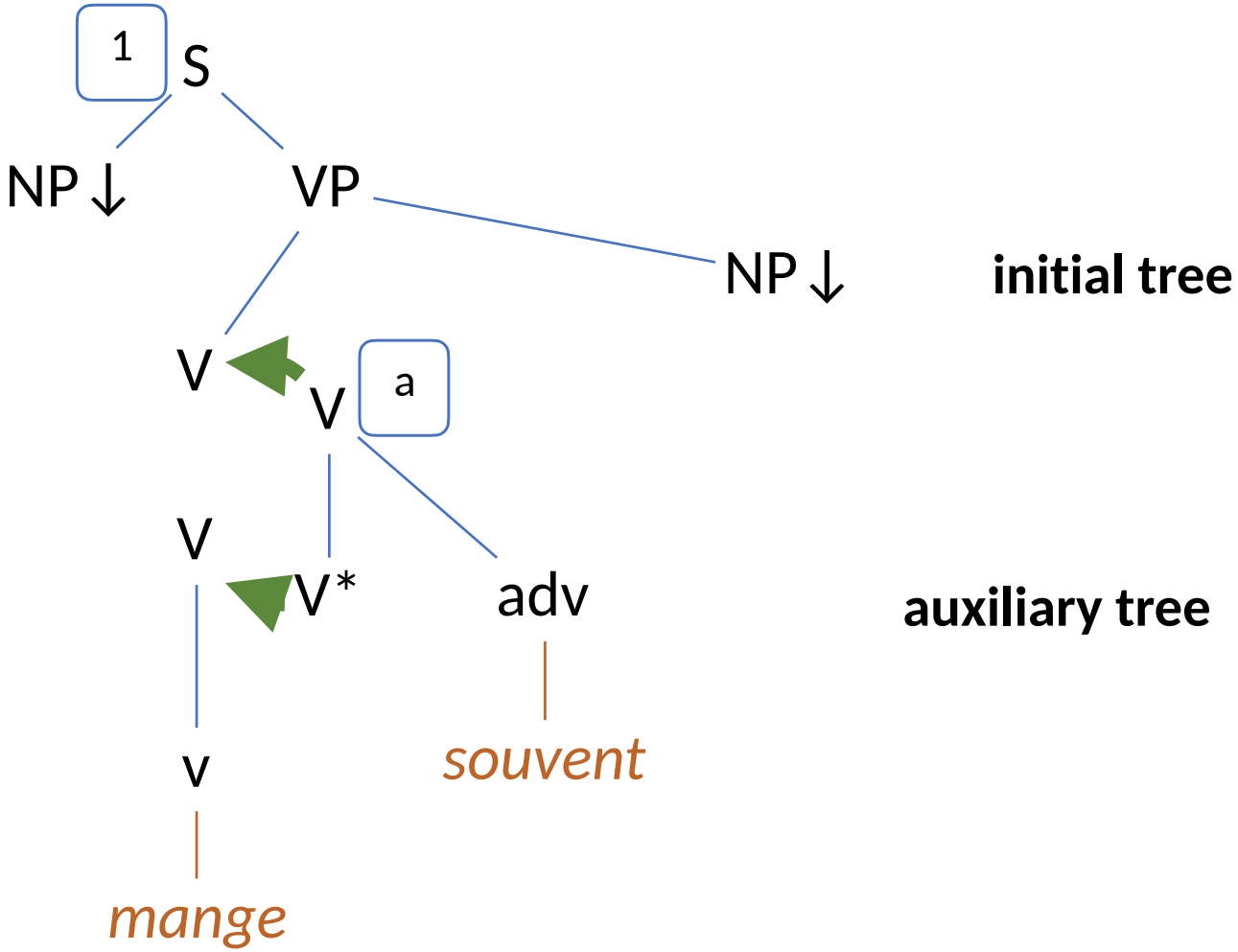
The adjunction operation



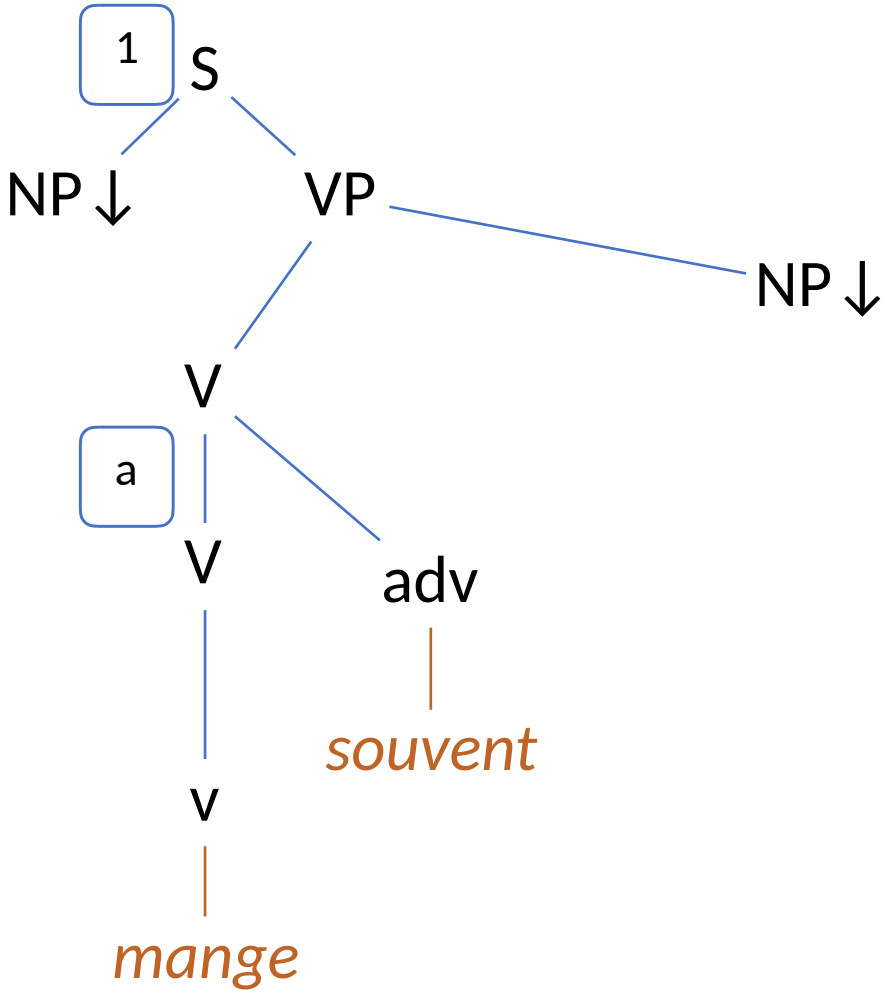
The adjunction operation



The adjunction operation

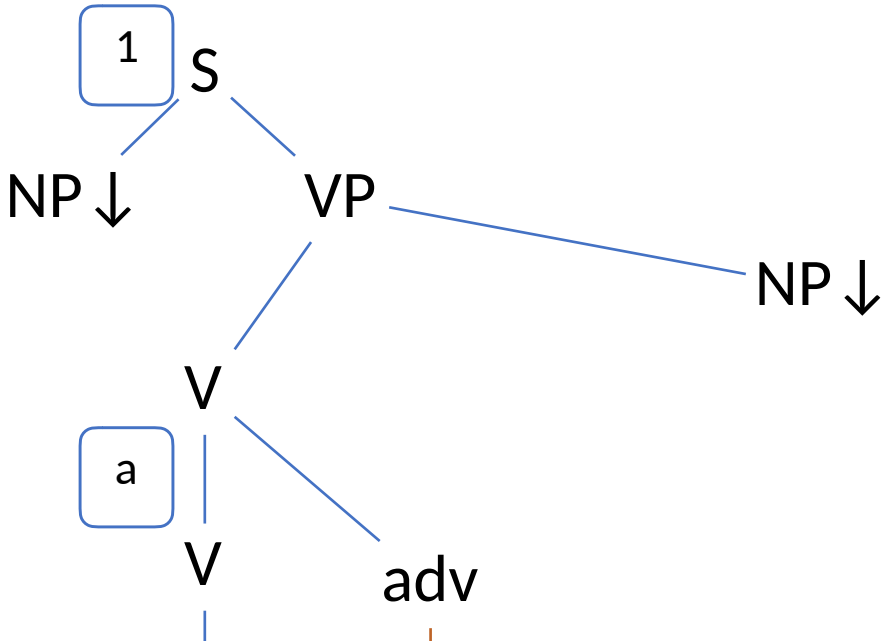


The adjunction operation



The adjunction operation

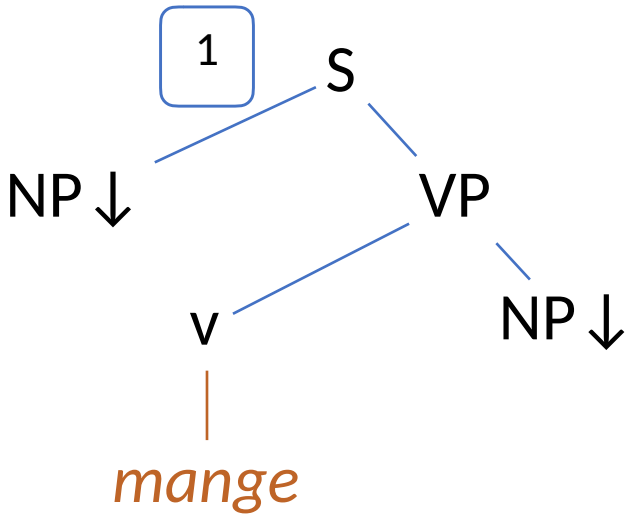
- In the derivation tree, we must now represent 2 distinct operations:
  - a substitution will be represented with a full line
  - an adjunction will be represented with a dashed line
- In the case of the previous example:

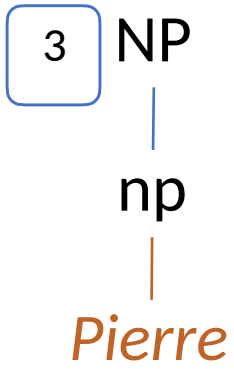


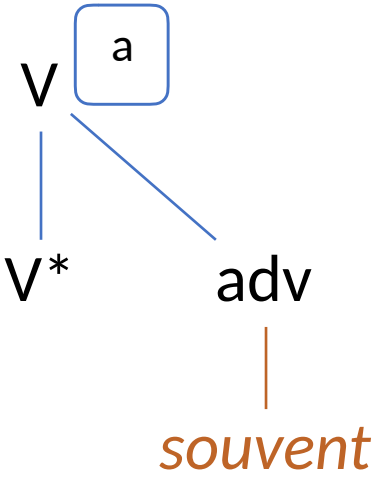
The adjunction operation

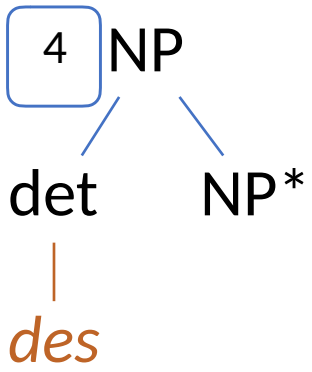
- In the derivation tree, we must now represent 2 distinct operations:
  - a substitution will be represented with a full line
  - an adjunction will be represented with a dashed line
- In the case of the previous example:

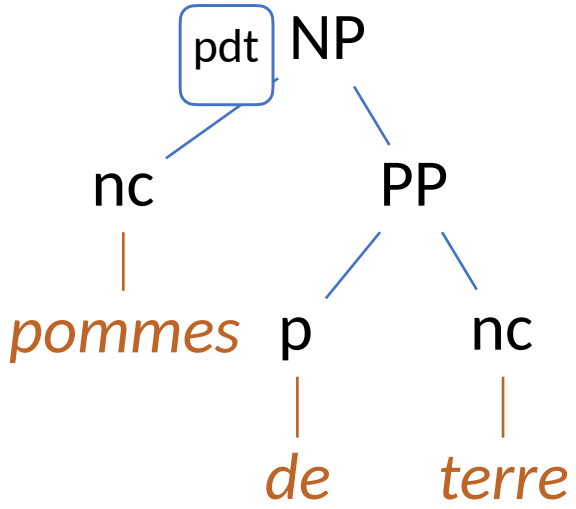
On our example





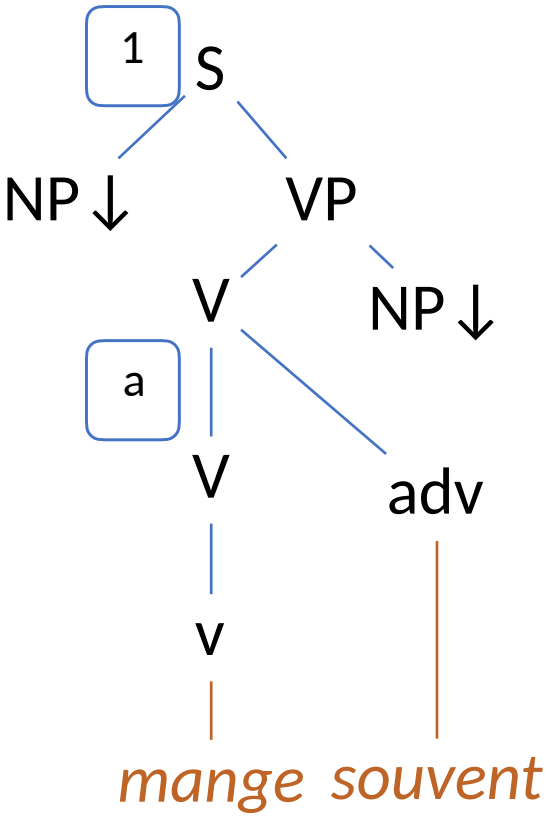


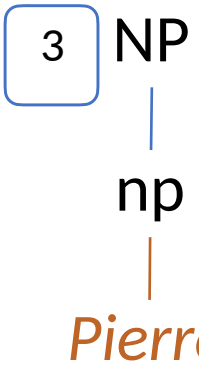


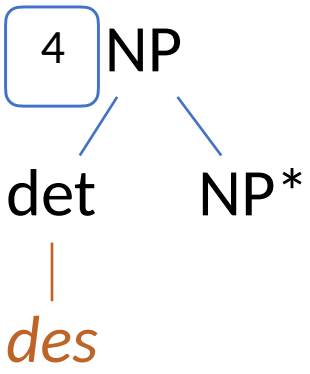


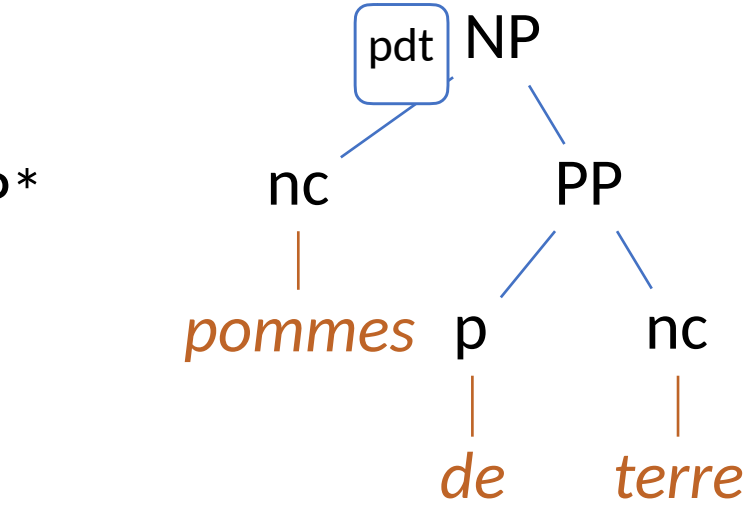
Derivation tree

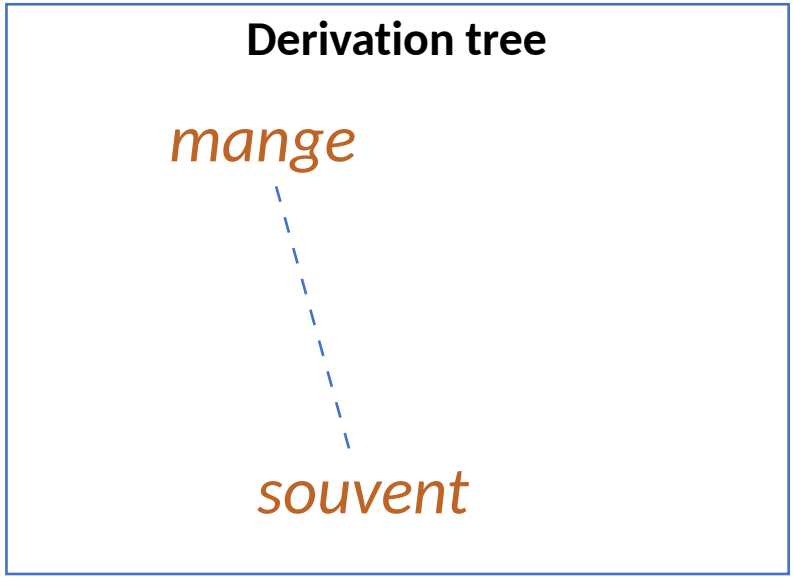
On our example



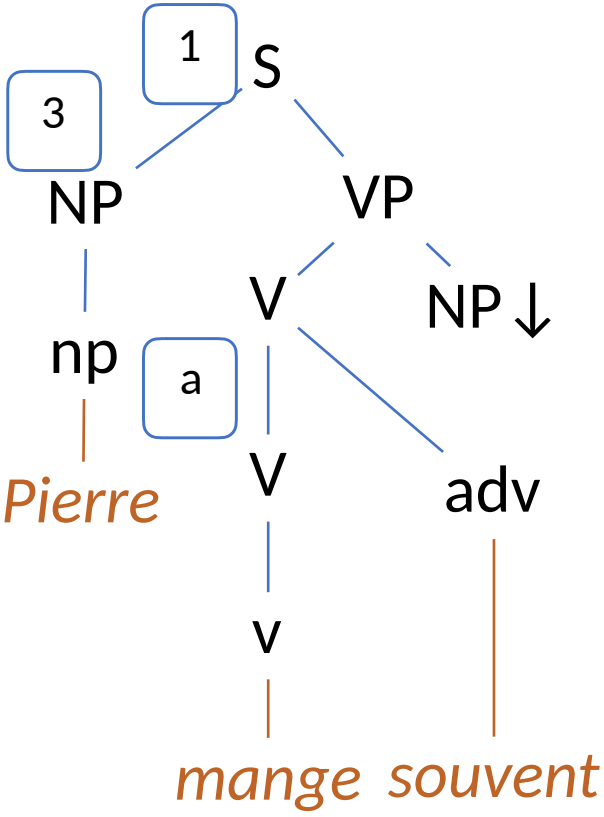


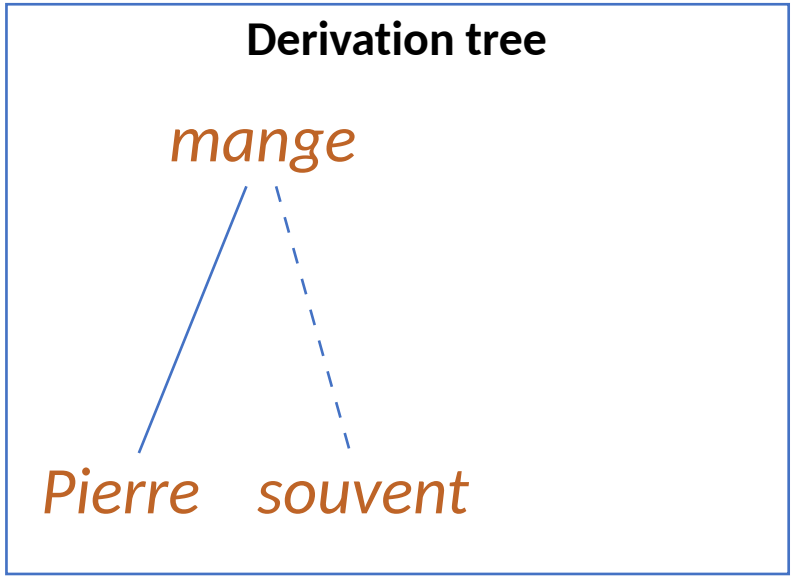


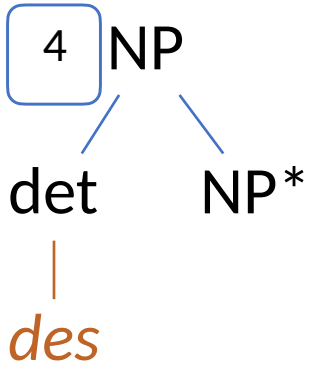


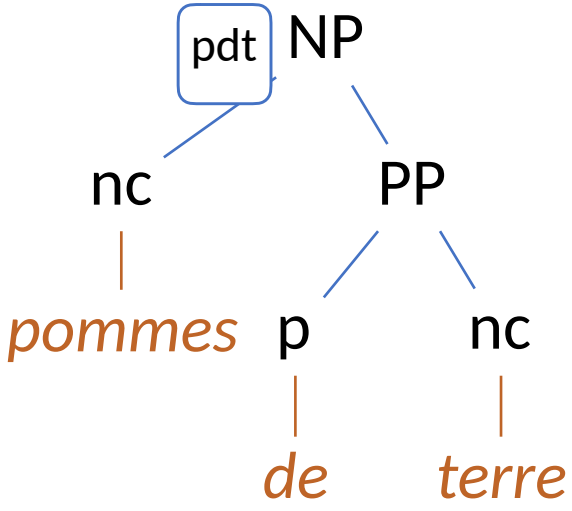


On our example

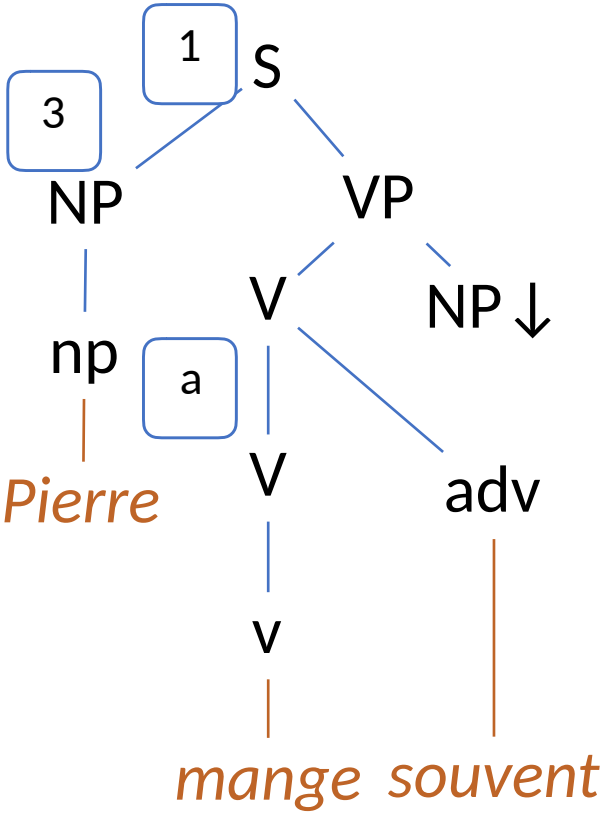


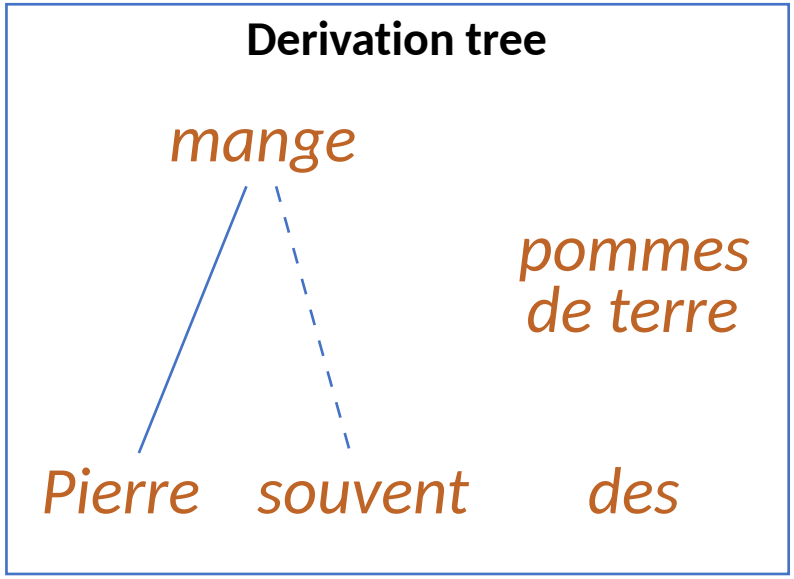


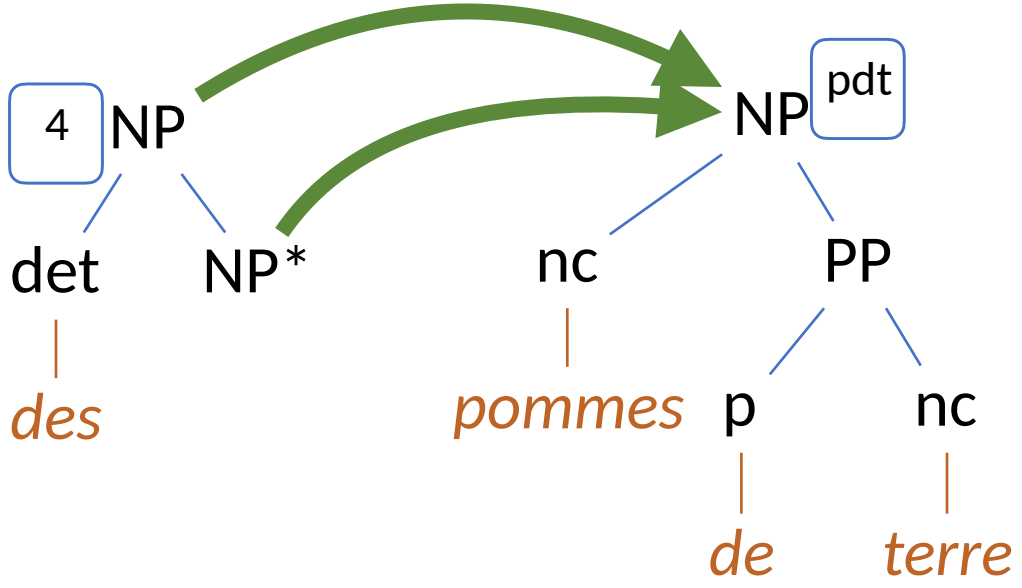




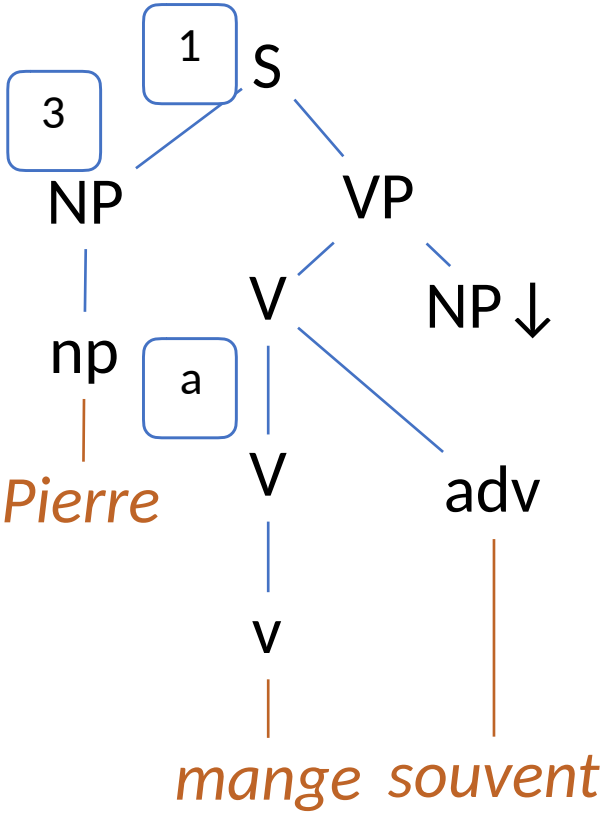
On our example

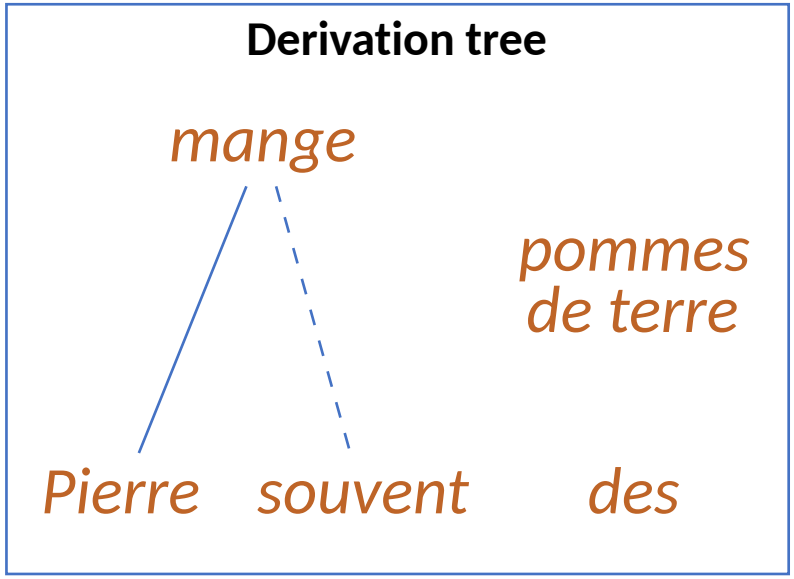


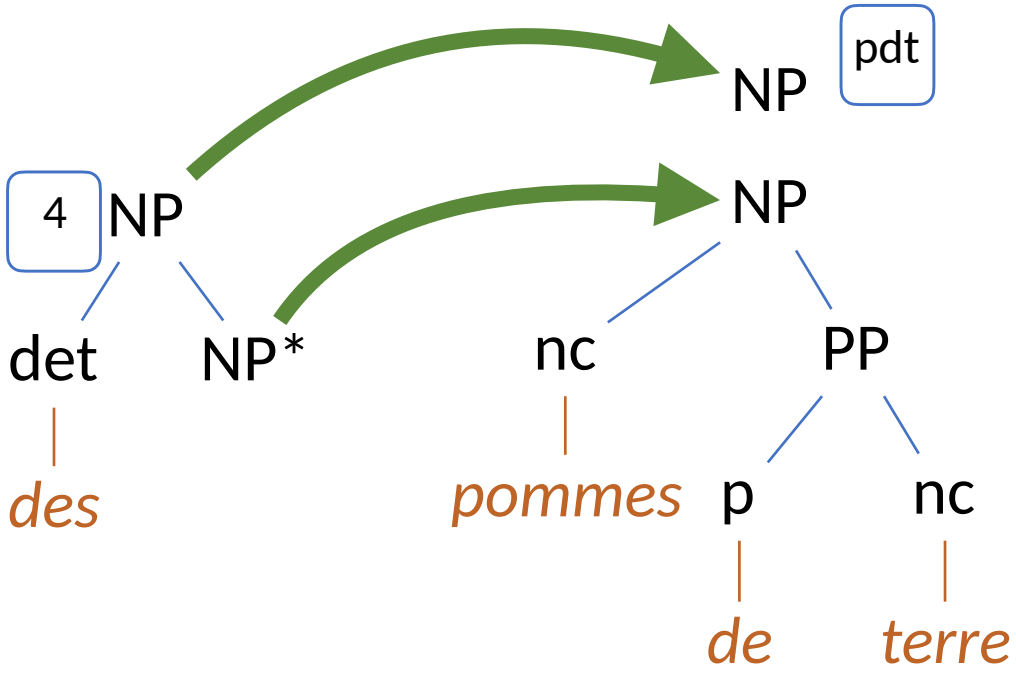




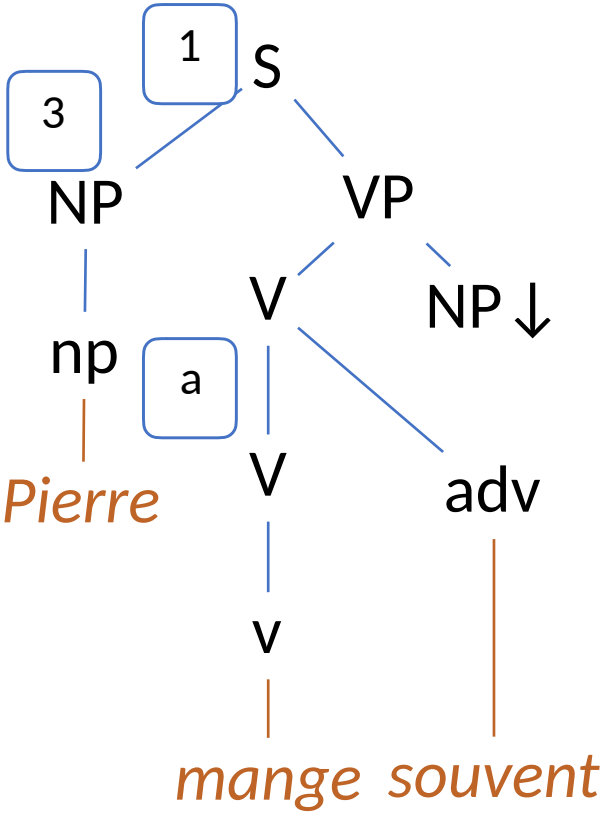
On our example

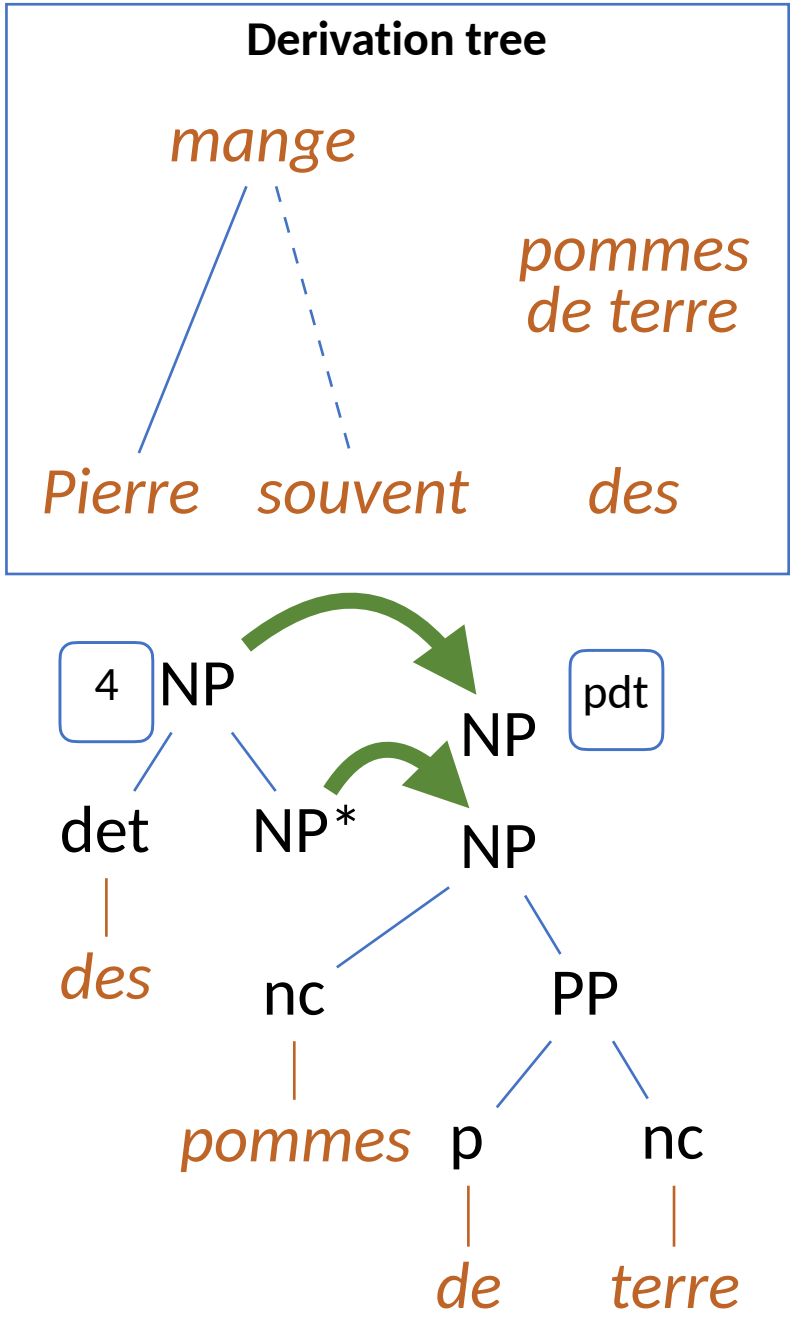




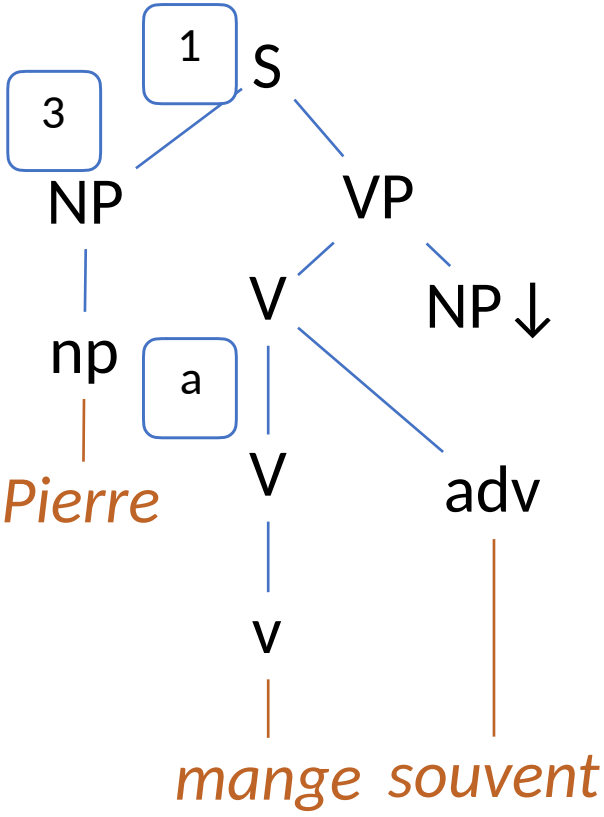


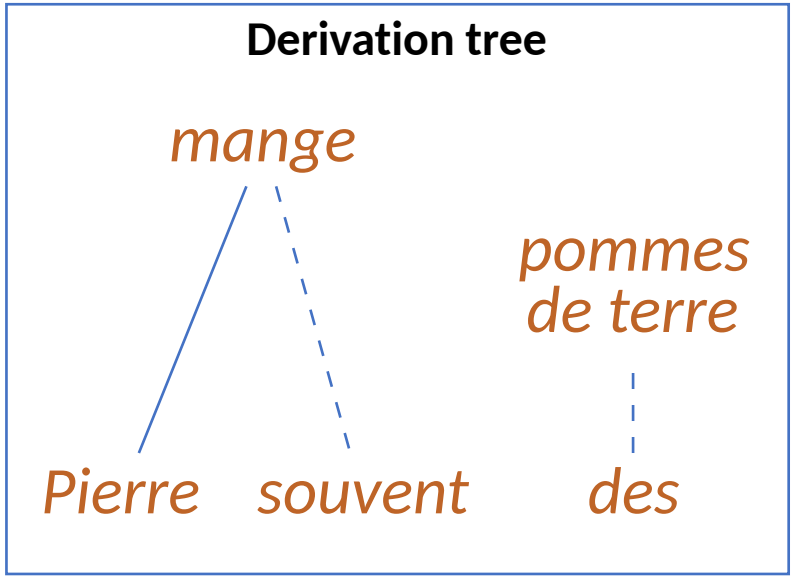
On our example

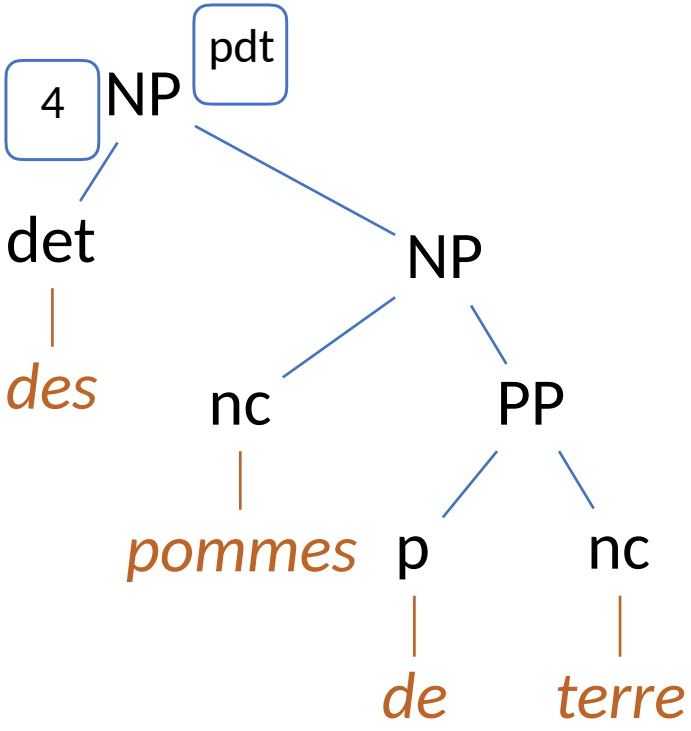




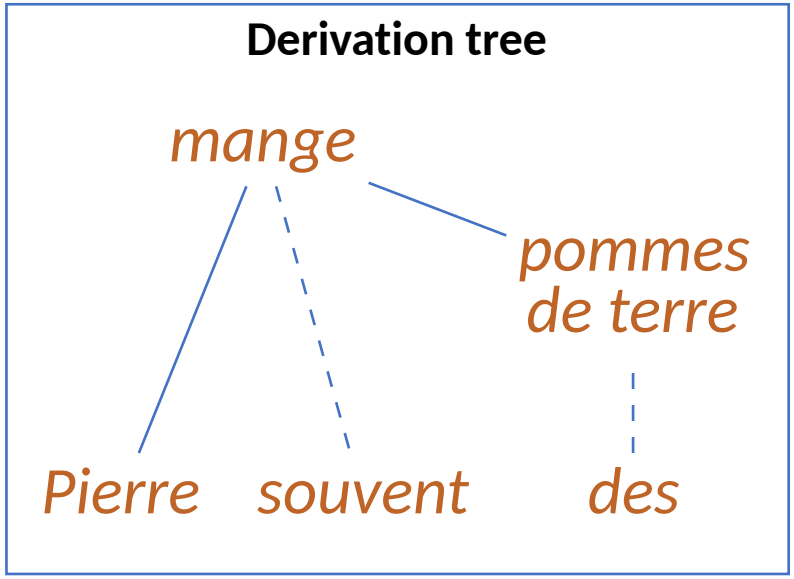
On our example

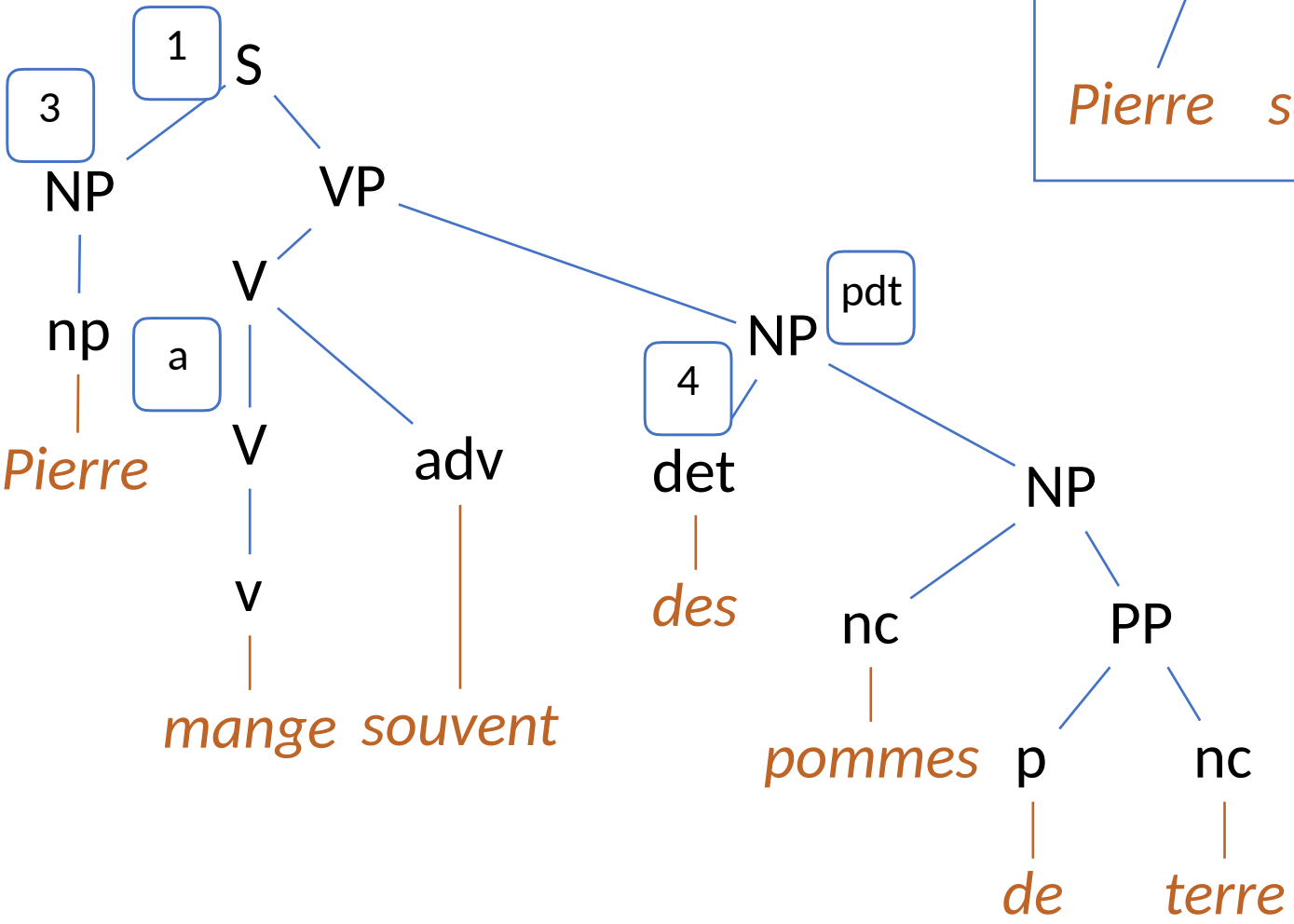






On our example



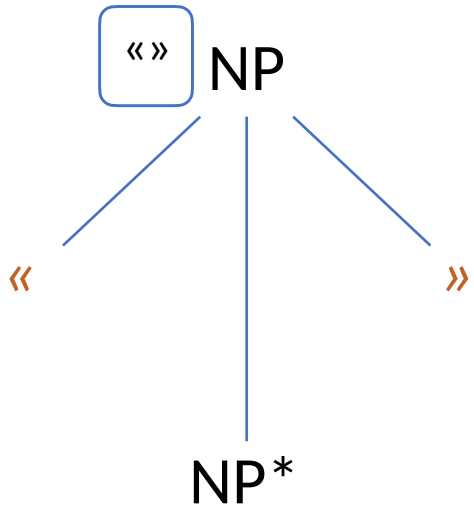


Wrapping trees

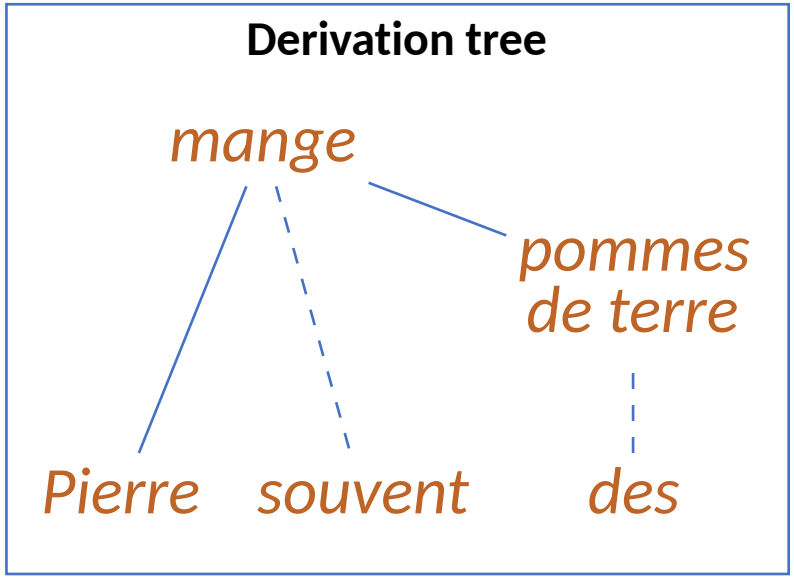
- There are no restrictions for auxiliary trees to have their spine on one side or the other of the tree
- TAGs allow **wrapping trees**

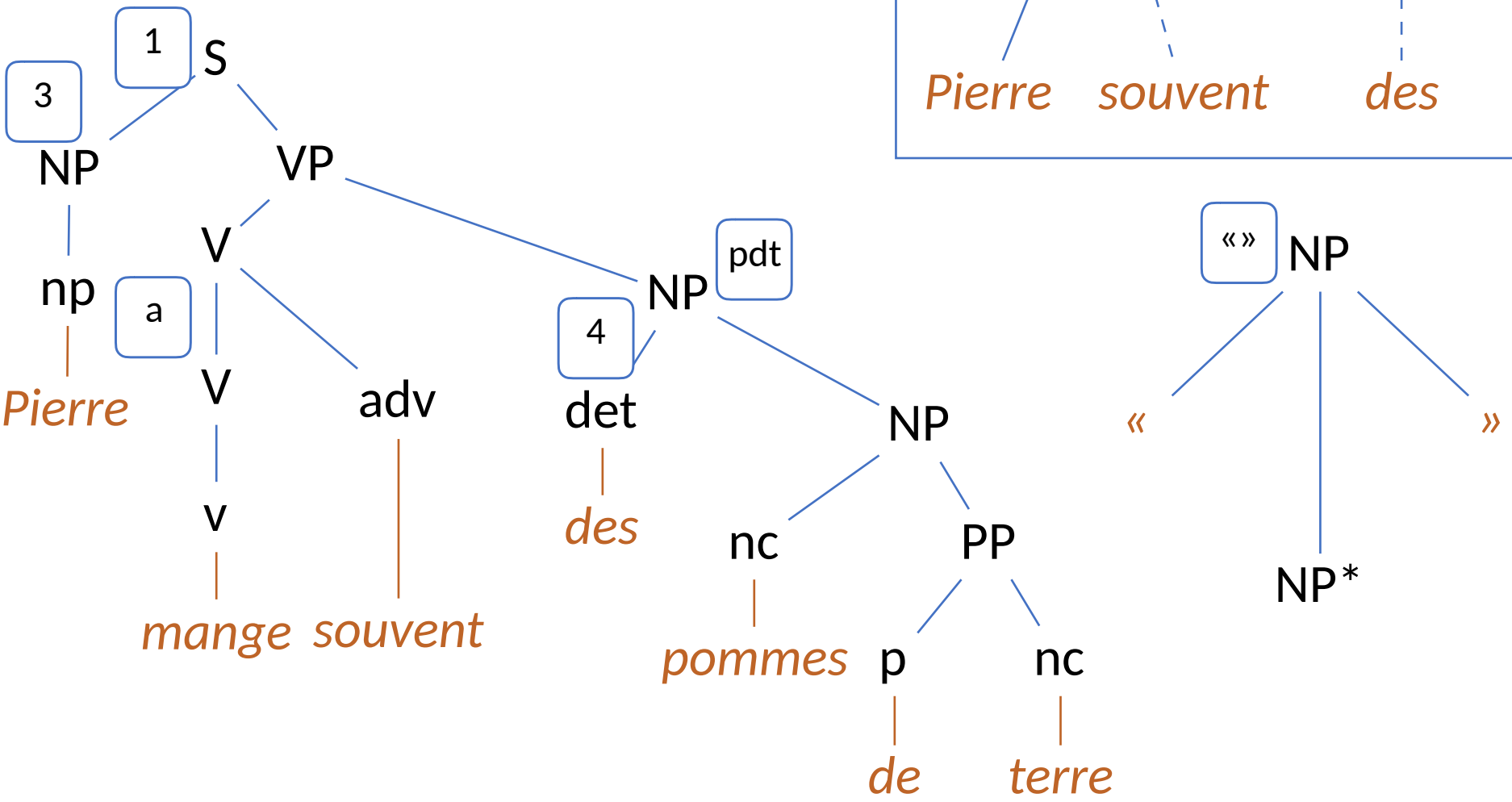
Wrapping trees

- There are no restrictions for auxiliary trees to have their spine one one side or the other of the tree
- TAGs allow **wrapping trees**

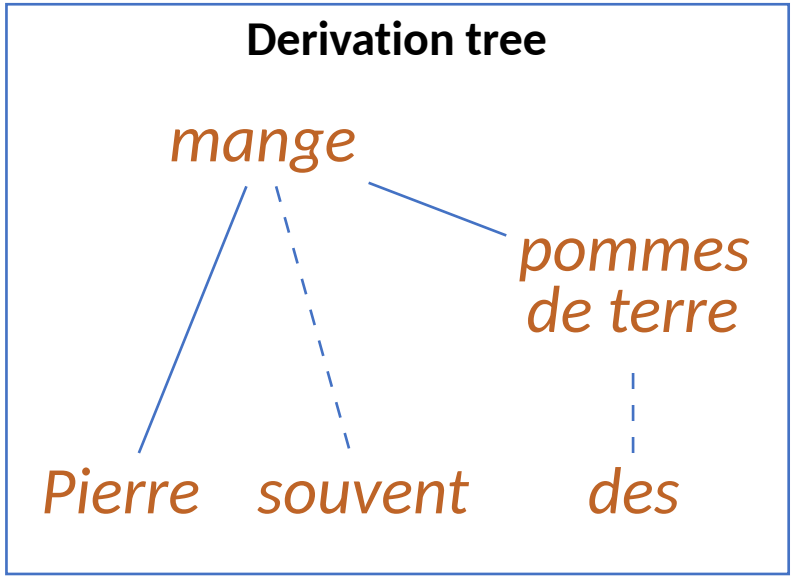


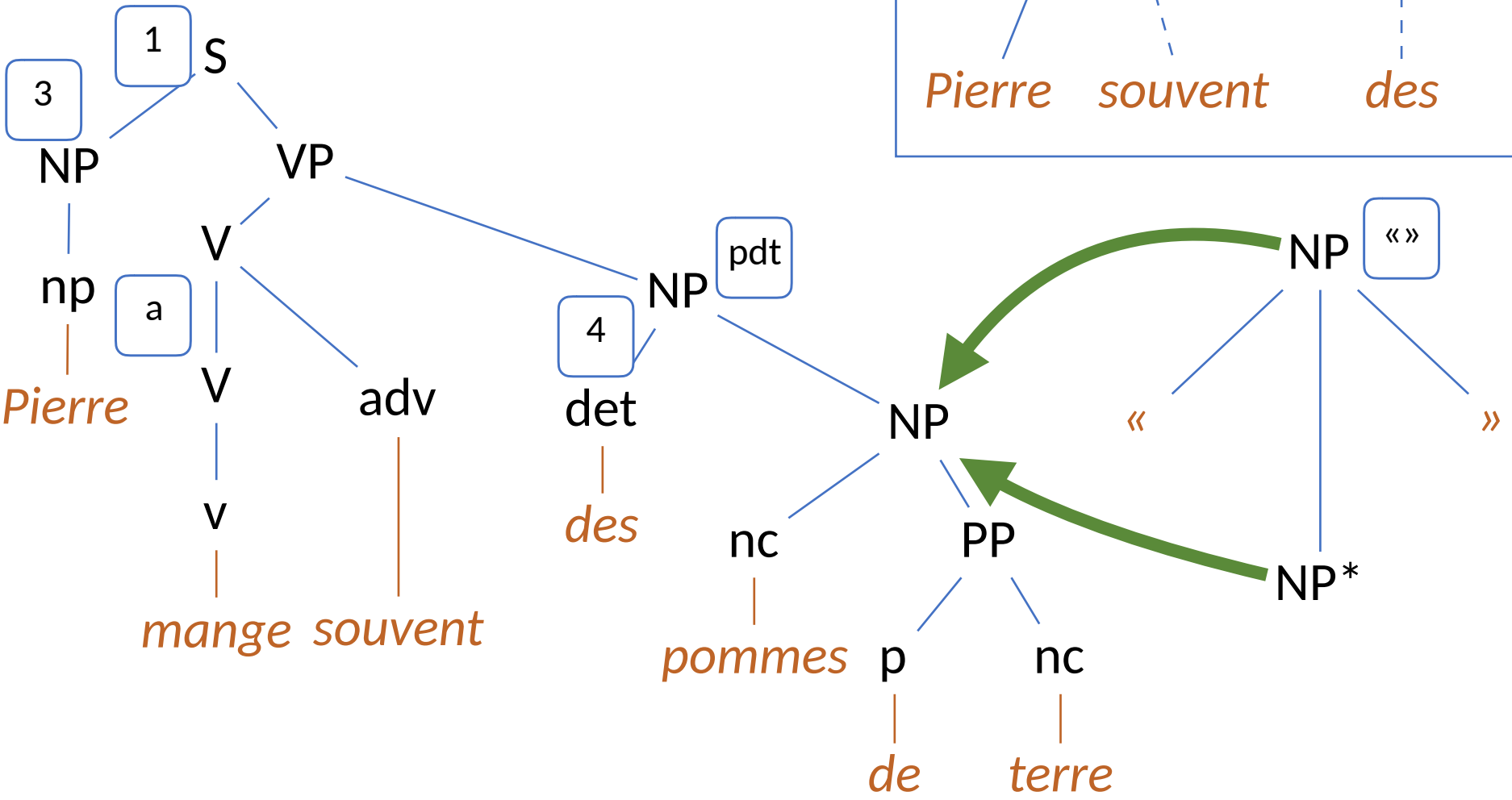
Wrapping trees



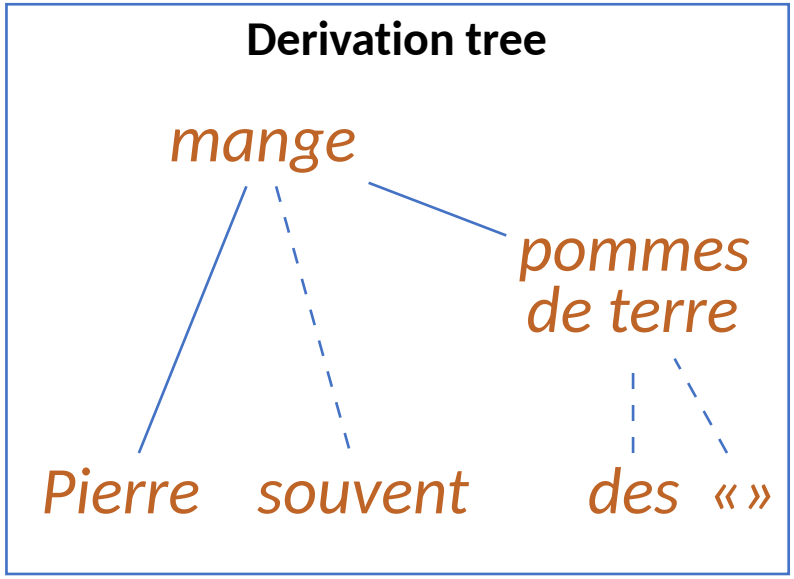


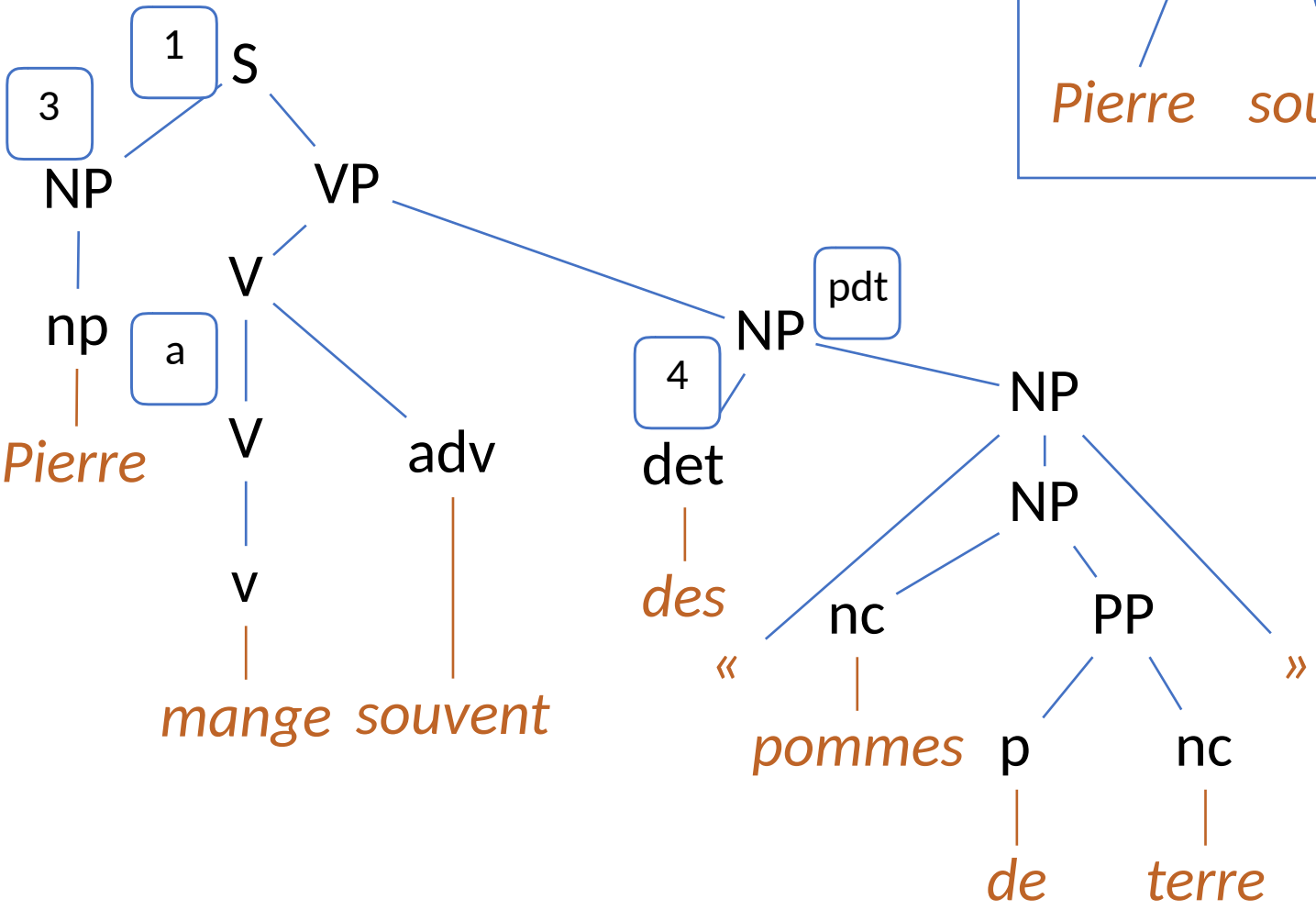
Wrapping trees





Wrapping trees





Expressive power of TAGs vs. CFGs

- The following context-free grammar defines $a^n b^n$:

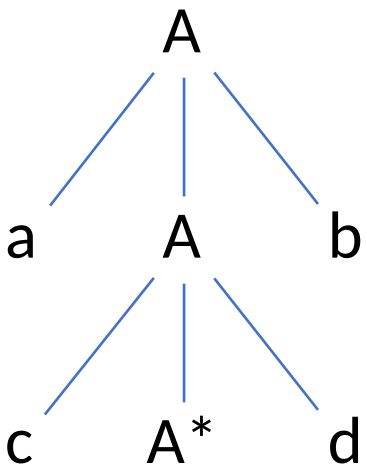
$$S \longrightarrow a\ S\ b$$

$$S \longrightarrow \varepsilon$$

- However, $\{a^n b^n c^n\}$ is not a context-free language
  - *a fortiori*, $\{a^n b^n c^n d^n\}$ is not a context-free language either
  - Proof: look for “pumping lemma for CFG” on Google
- 2-copy is not a context-free language either

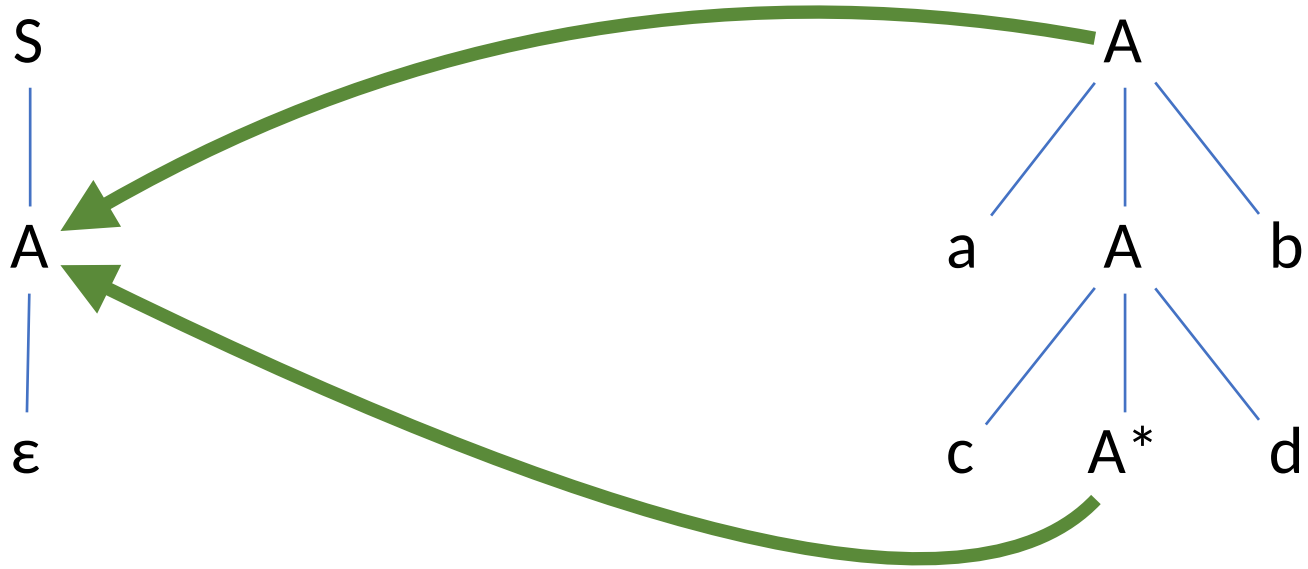
Expressive power of TAGs vs. CFGs

- Wrapping tree give an extra expressive power



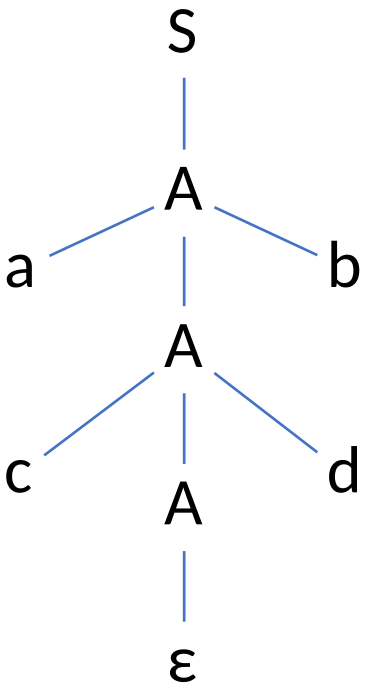
Expressive power of TAGs vs. CFGs

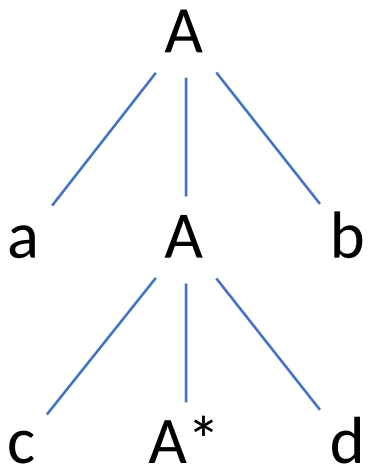
- Wrapping tree give an extra expressive power



Expressive power of TAGs vs. CFGs

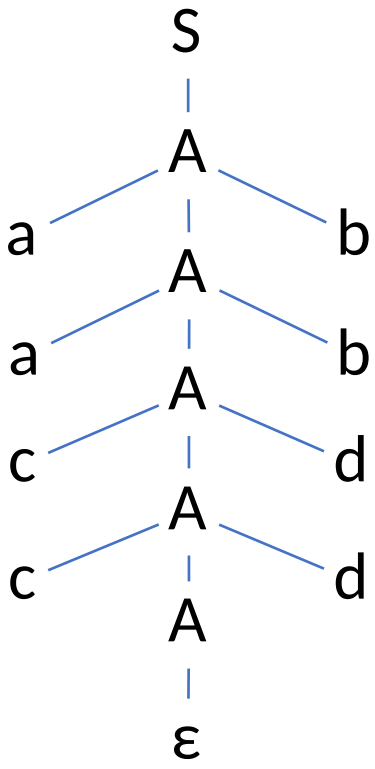
- Wrapping tree give an extra expressive power

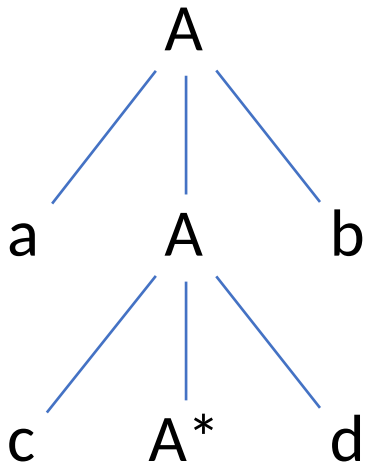




Expressive power of TAGs vs. CFGs

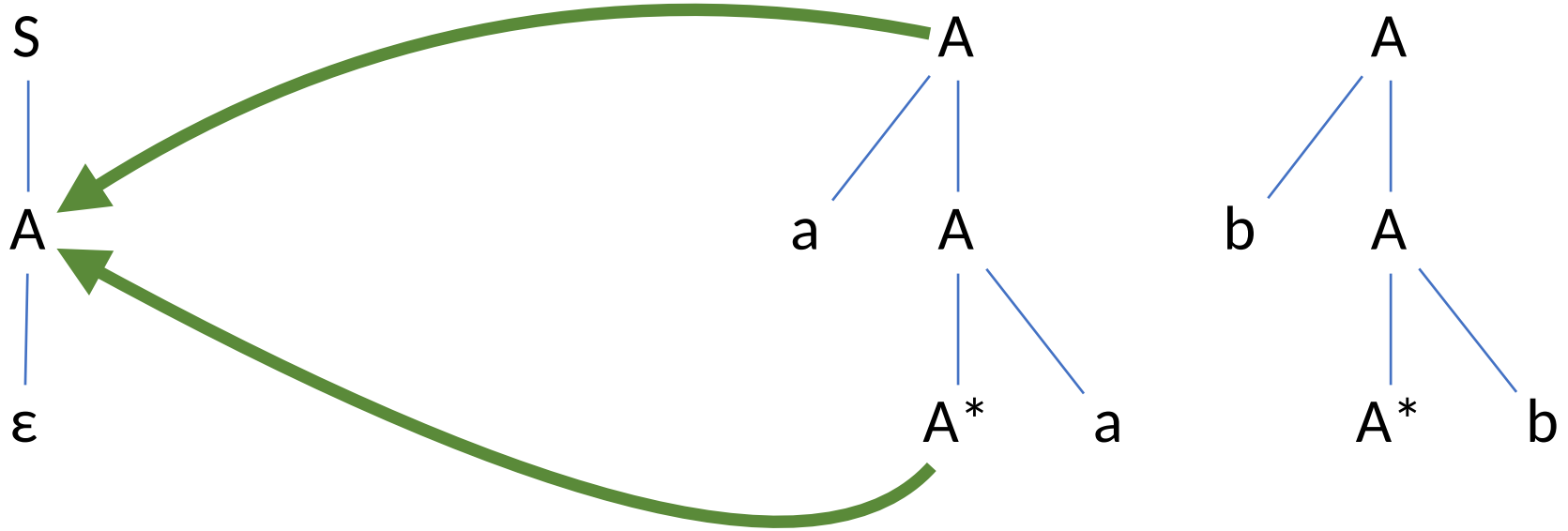
- Wrapping tree give an extra expressive power
  - There are in fact additional details to make this example fully work, but the underlying idea is the right one





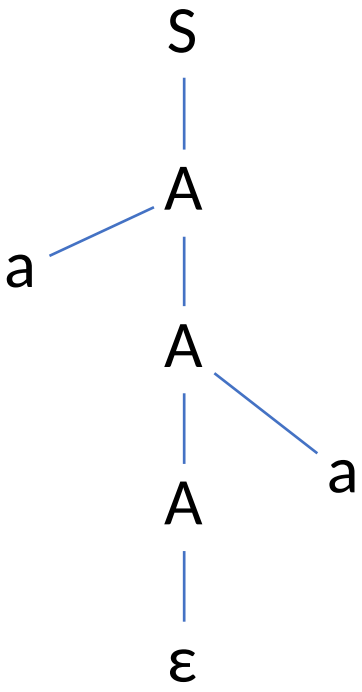
Expressive power of TAGs vs. CFGs

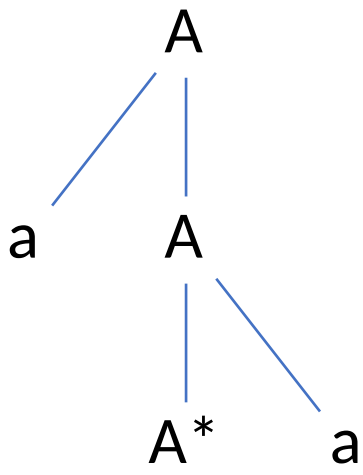
- Wrapping tree give an extra expressive power

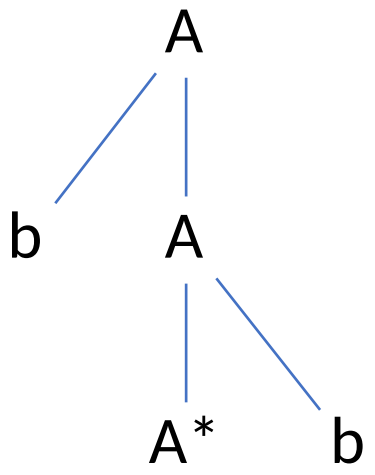


Expressive power of TAGs vs. CFGs

- Wrapping tree give an extra expressive power

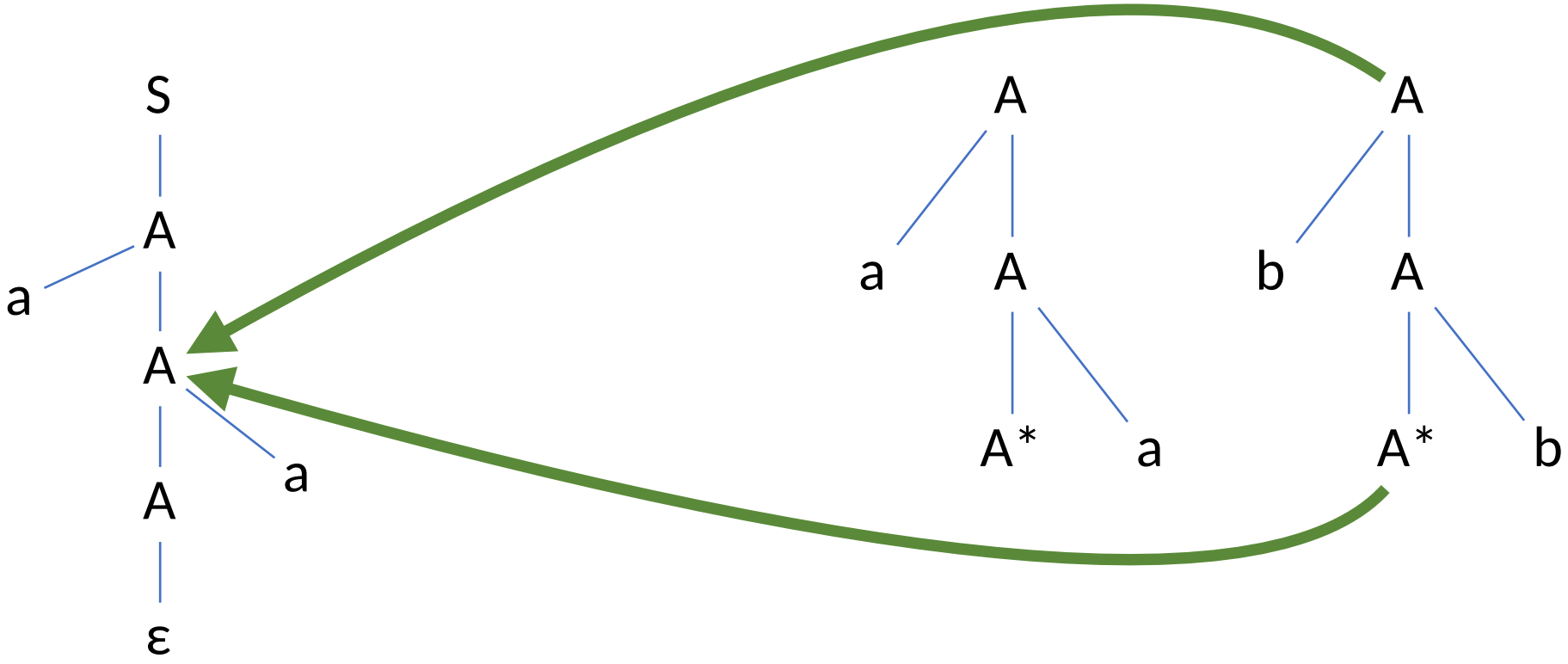






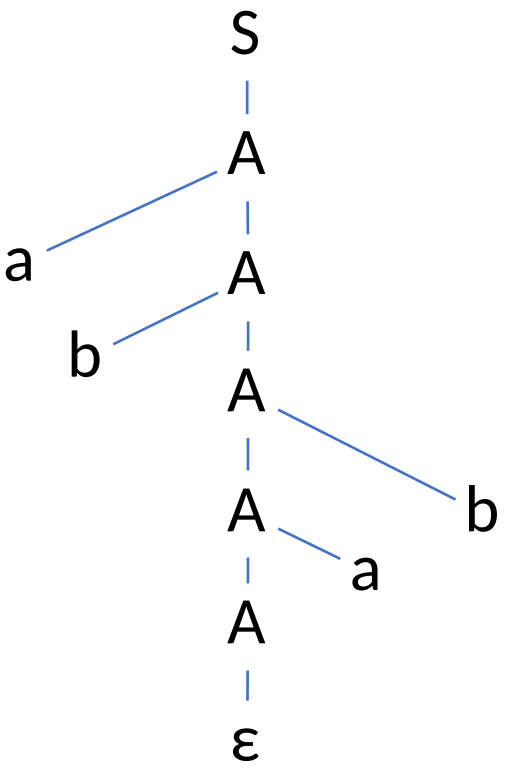
Expressive power of TAGs vs. CFGs

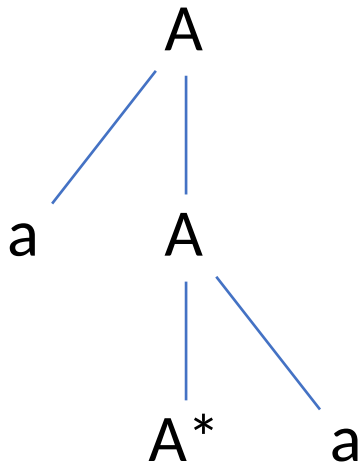
- Wrapping tree give an extra expressive power

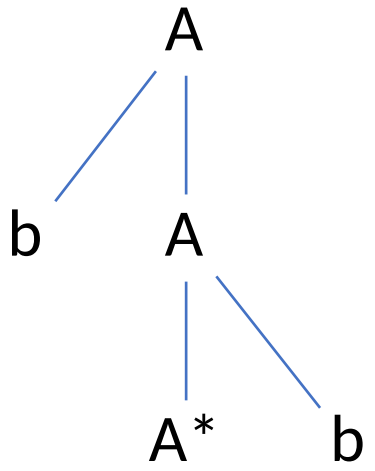


Expressive power of TAGs vs. CFGs

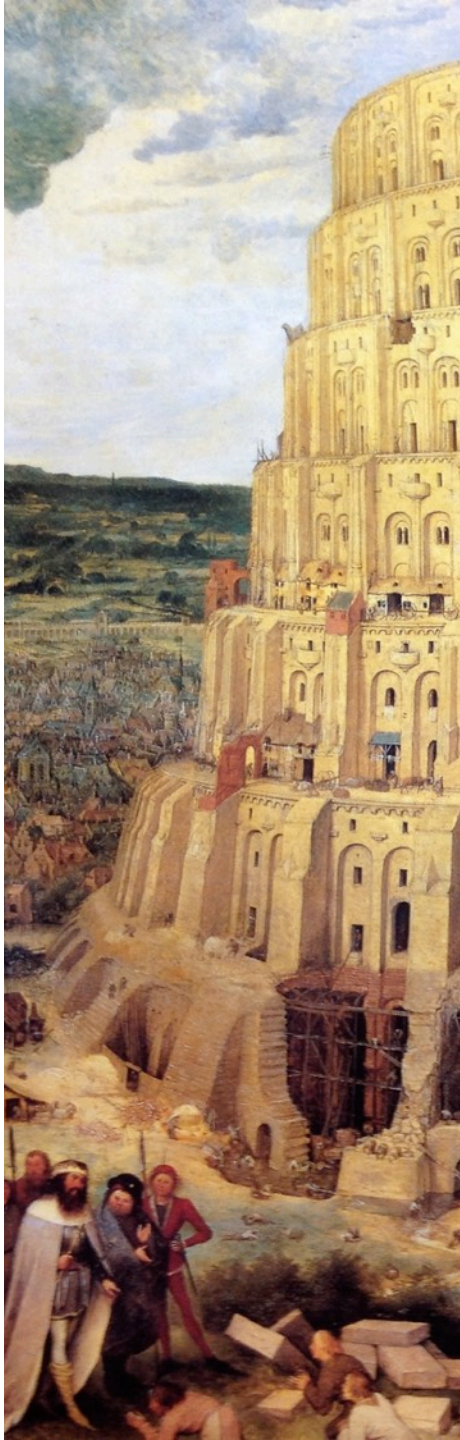
- Wrapping tree give an extra expressive power







Complexity intermezzo



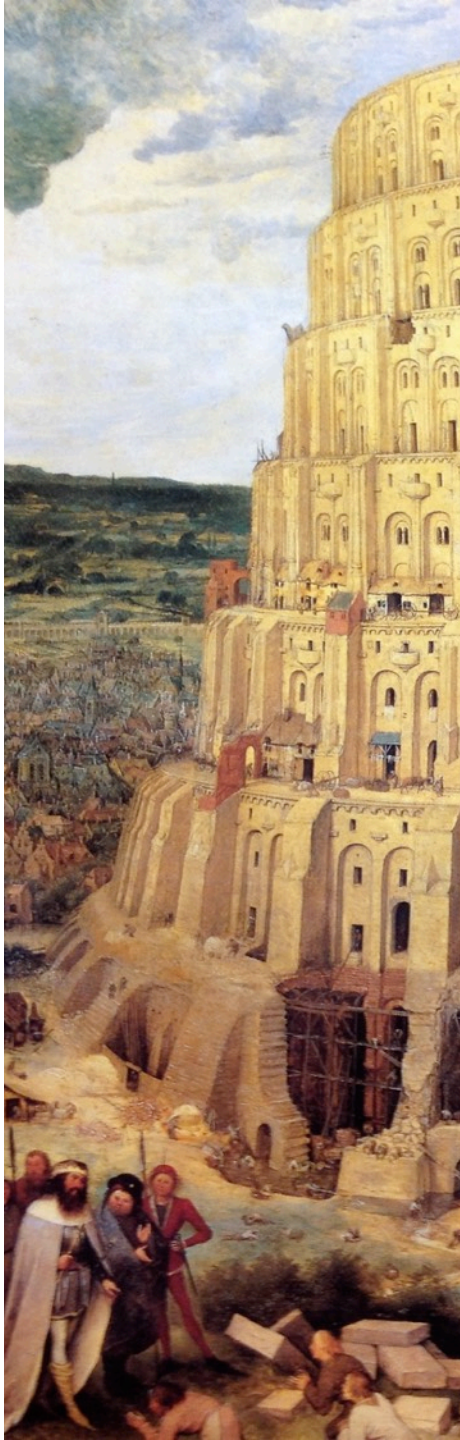
Parsing complexity

- **CFGs and TSDs** are parsable in cubic time, i.e. $O(n^3)$, where n is the length of the input sentence
- **TAGs**, in the general case, are parsable in $O(n^6)$
  - Tree Adjoining Languages are closed under union, concatenation, and Kleene-star
- Proofs for these complexities assume binary elementary trees
  - There are trivial ways to turn a non-binary tree into a strictly equivalent binary tree
  - For CFGs, there is an even more restrictive form in which any grammar can be converted, the *Chomsky normal form*
    - Rules of the form $A \rightarrow BC$ or $A \rightarrow a$ or $S \rightarrow \varepsilon$ (S is the axiom)

Parsing complexity

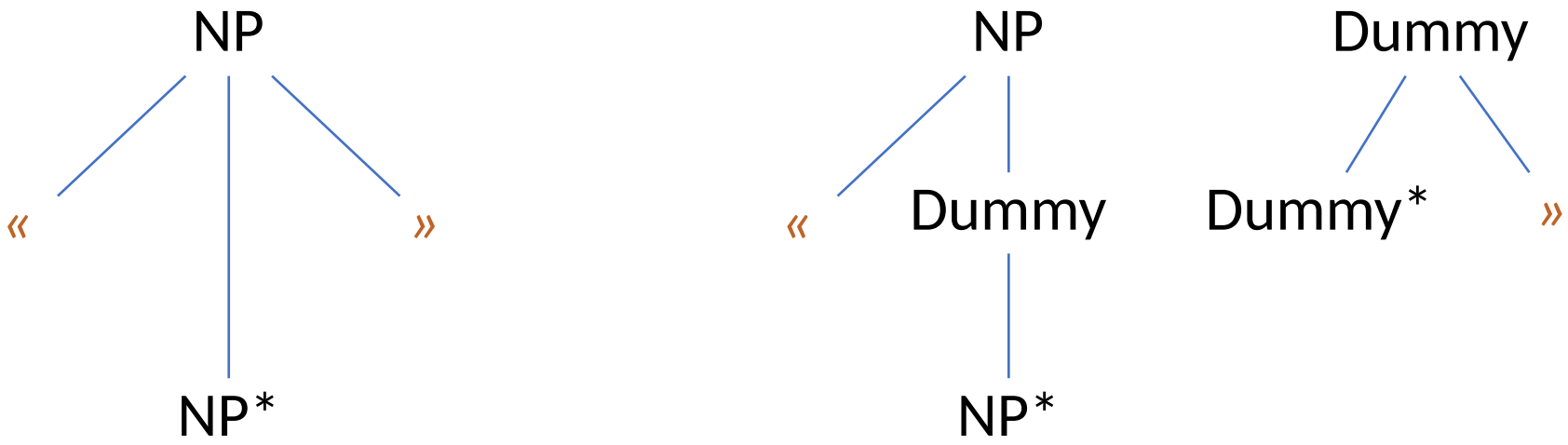
- However, the adjunction operation is a useful addition
- Can we keep the adjunction operation without the additional complexity
  - YES: we just need to get rid of wrapping adjunctions
    > Tree Insertion Grammars

Tree insertion grammars



Getting rid of wrapping adjunctions

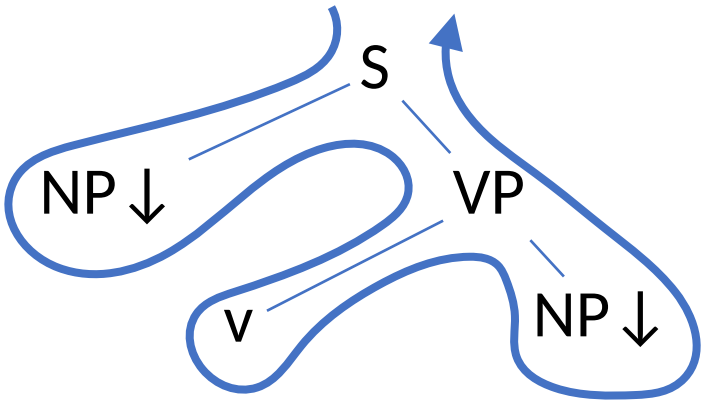
- Underlying intuition:
  - we need to forbid wrapping auxiliary trees
  - we also need to forbid any way to simulate wrapping

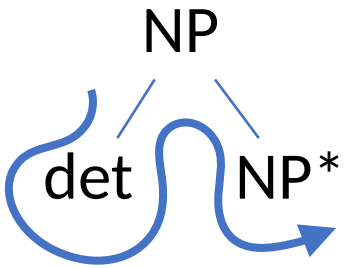


Tree insertion grammars

- A Tree Insertion Grammar (TIG) is a TAG where wrapping adjunction is forbidden
- TIG are parsable in $O(n^3)$!
  - We will prove it by showing how it is possible to convert a TIG into a CFG
  - The underlying intuition is to rewrite each elementary tree in the form of an equivalent CFG rewriting rule
  - using a top-down traversal of the trees

Converting a TIG into a CFG





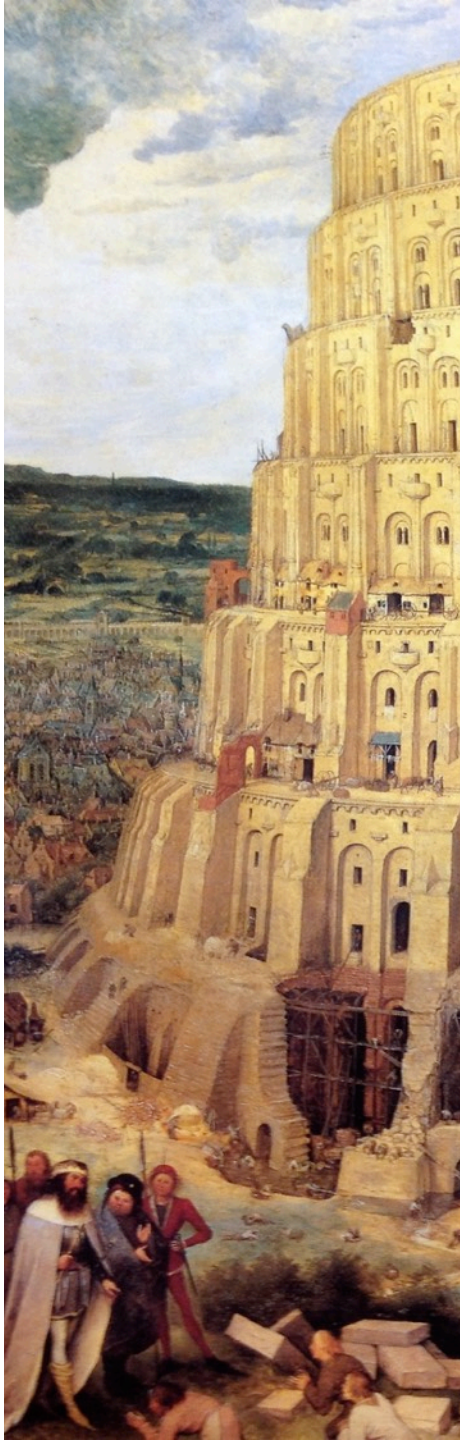
$S \longrightarrow NP \;\; VP_g \;\; v \;\; NP \;\; VP_d$

$NP_g \longrightarrow det \; NP_g$

TAGs and TIGs in practice?

- In practice, developing a TAG (or a TIG) is a very time-consuming task
- Two major ways out
  - Use a more abstract way to represent grammar rules ("metagrammars") that can generate a TAG (or a TIG)
    - This is how one of the best performing parsers for French was developed (FRMG, de La Clergerie et al.)
  - Extract it from a treebank
    - This allows for integrating probabilities, following the standard way to probabilise CFGs

Probabilistic CFGs

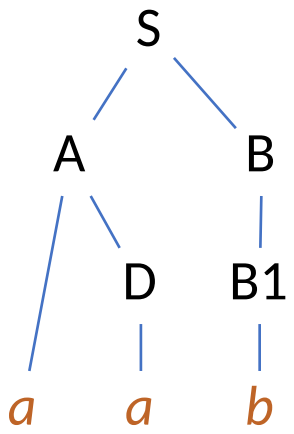


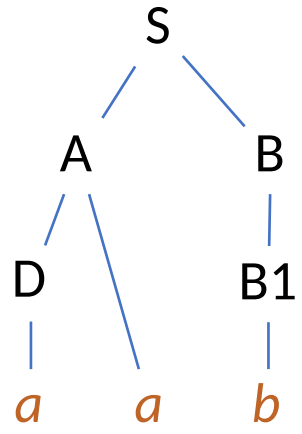
PCFGs

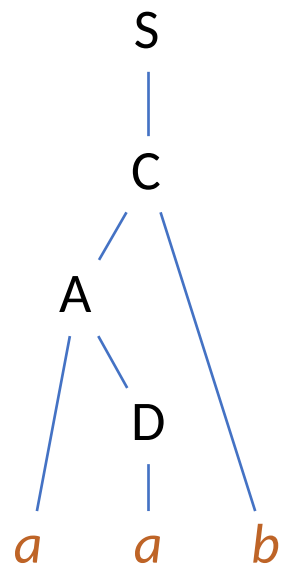
- Probabilistic CFGs (PCFGs) are a direct extension of CFGs
- The probabilisation of a grammatical formalism consists in defining a way to assign a probability to each sentence in the language generated by the grammar
  - Of course, the (often infinite) sum of to probabilities for all sentences in the language must be =1
- CFGs are **non-contextual**
  - A reasonable way to assign a probability distribution over CFGs is to do so in a **non-contextual way**
  - In other words, probabilities will be associated with each rewriting rule
  - The probability of a sentence according to the grammar will be the product of the probabilities of all rewritings in the derivation

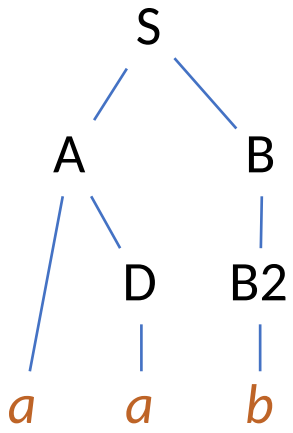
PCFG on an example

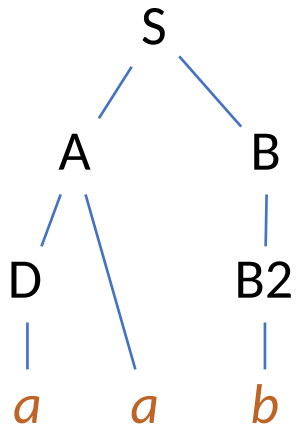
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

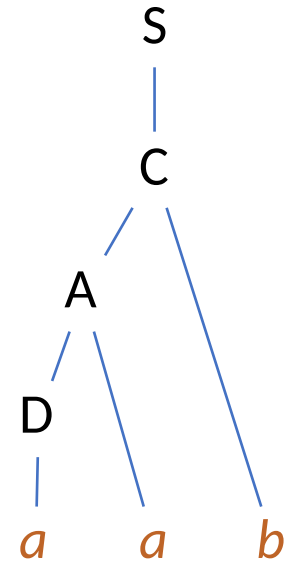








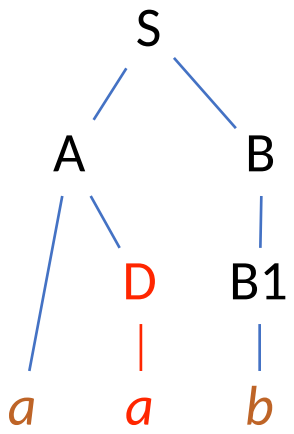


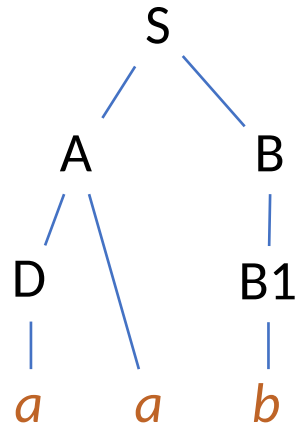


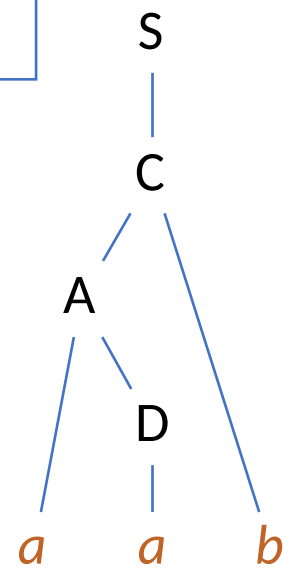
PCFG on an example

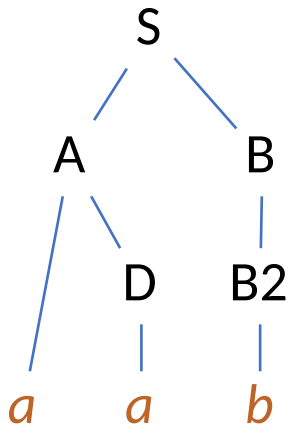
1

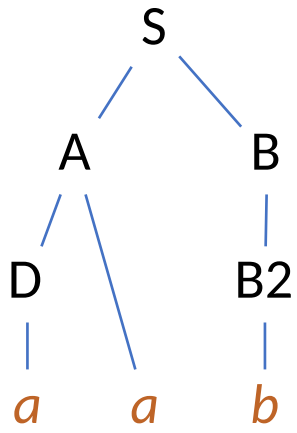
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

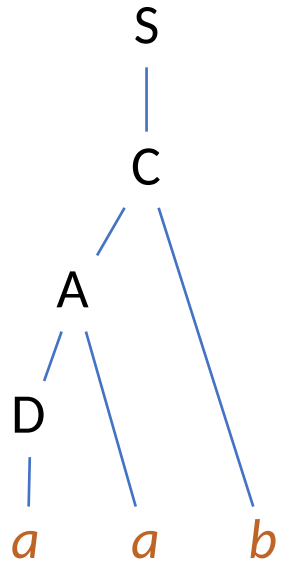








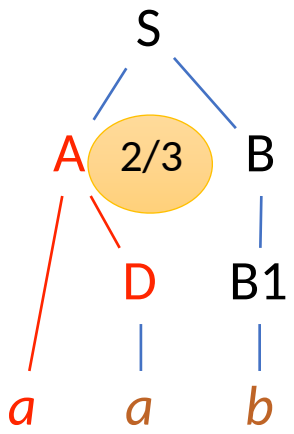


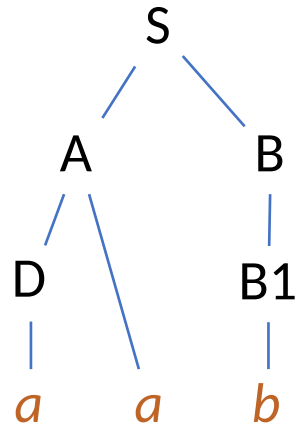


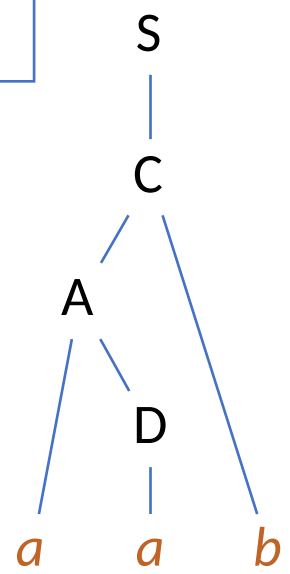
PCFG on an example

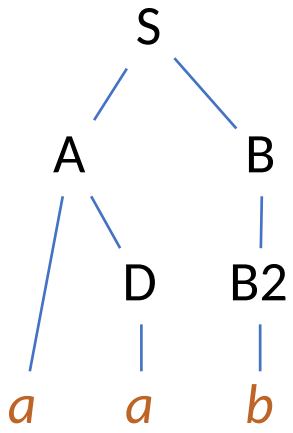
(1*2/3)

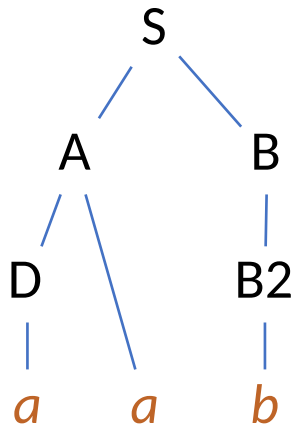
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

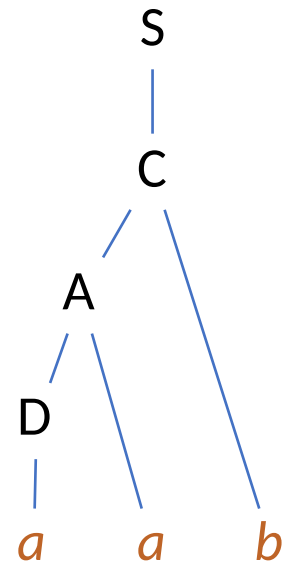








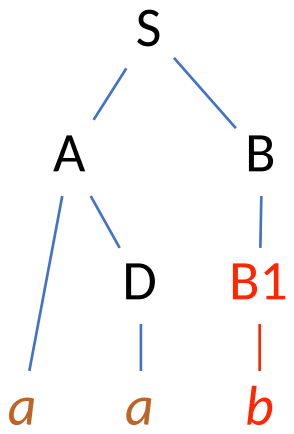


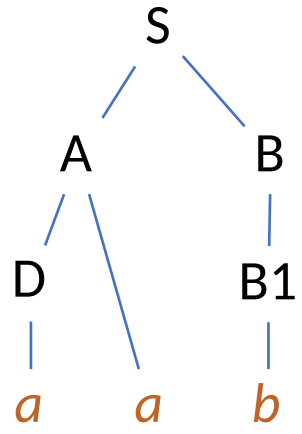


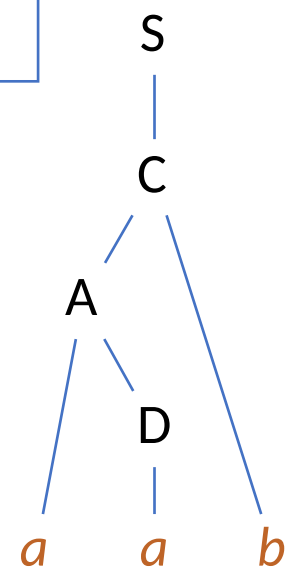
PCFG on an example

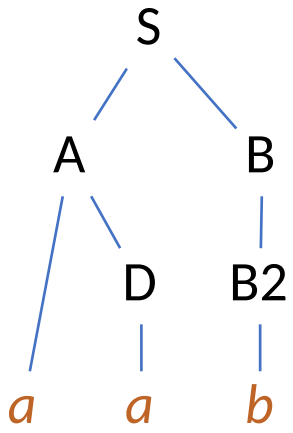
(1*2/3) * (1

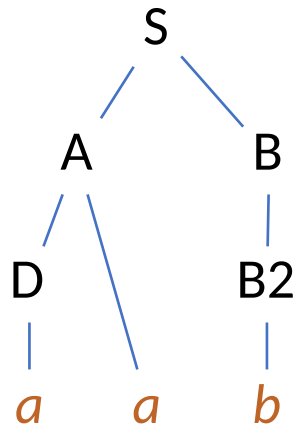
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

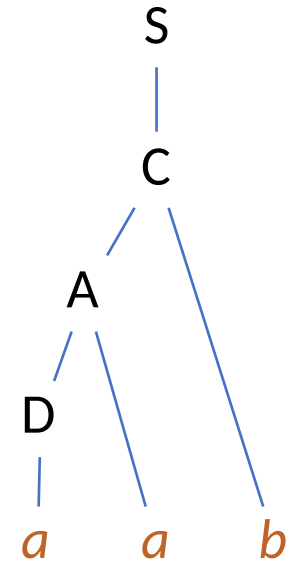








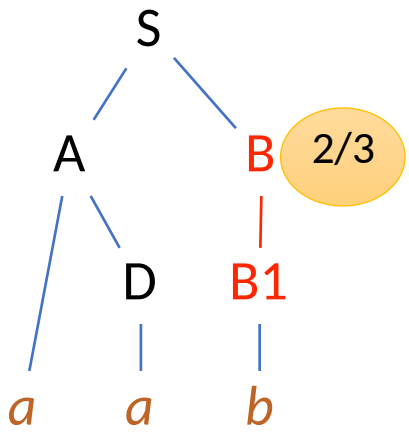


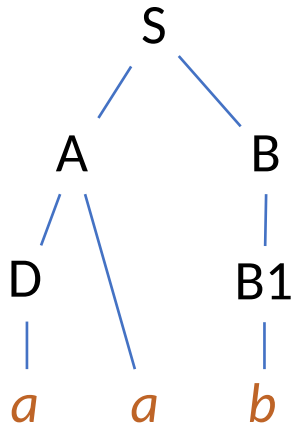


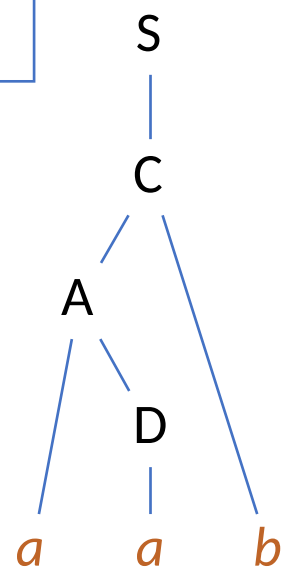
PCFG on an example

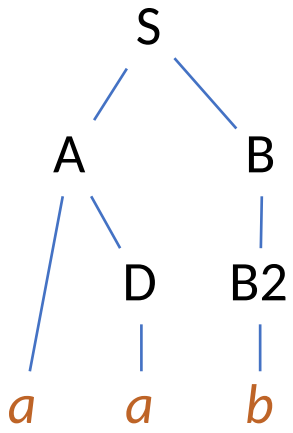
(1*2/3) * (1*2/3)

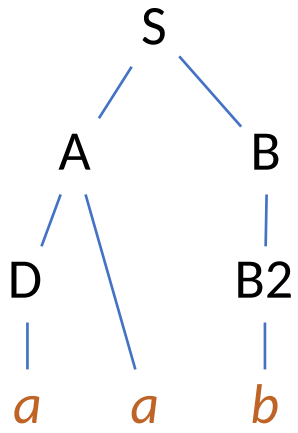
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

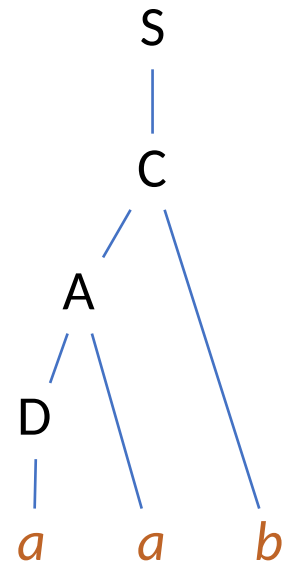








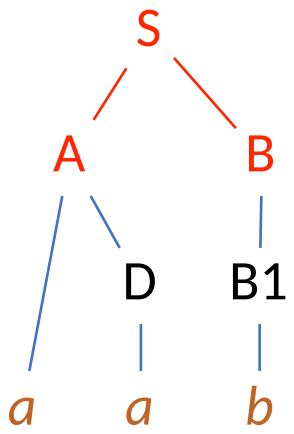


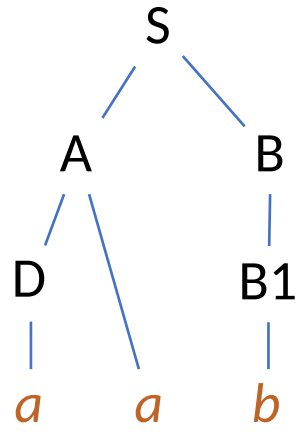


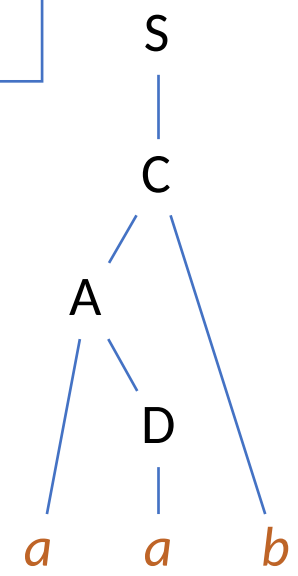
PCFG on an example

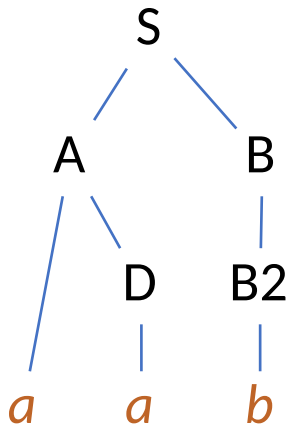
(1*2/3) * (1*2/3) * 3/4

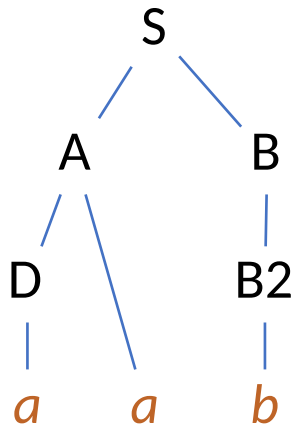
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

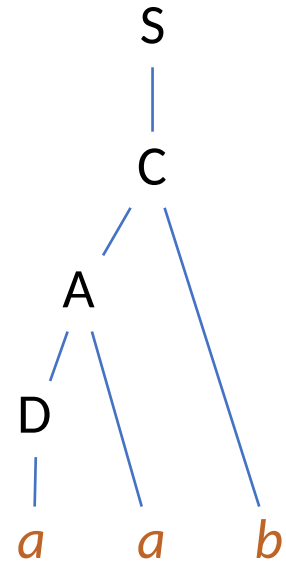








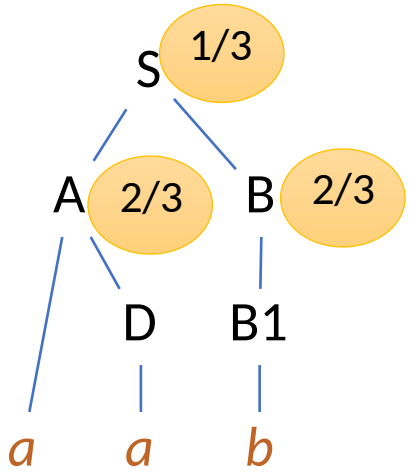


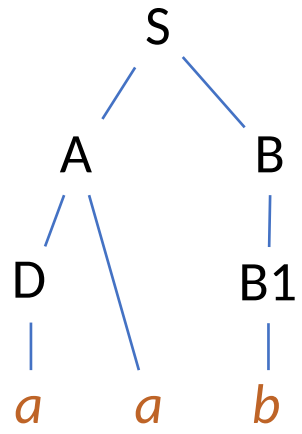


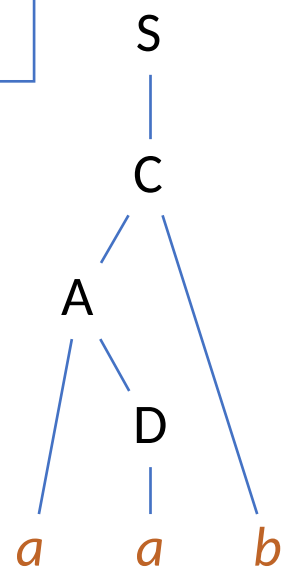
PCFG on an example

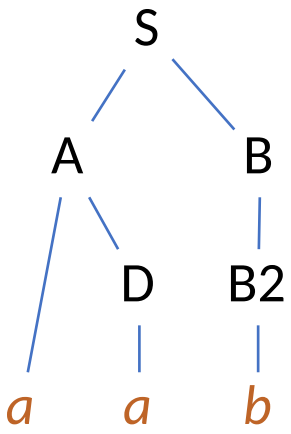
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

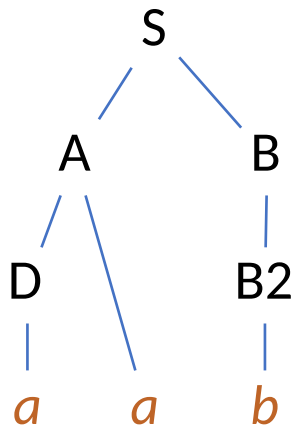
(1*2/3) * (1*2/3) * 3/4 = 1/3

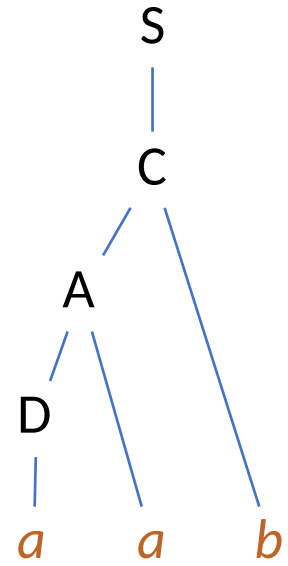






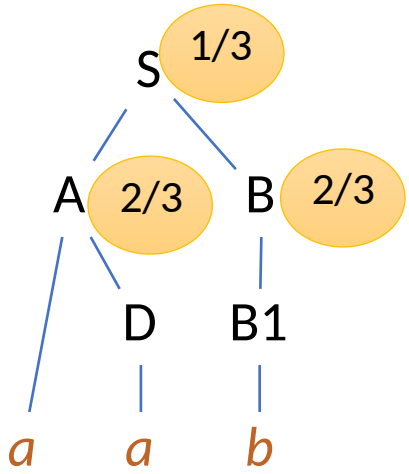


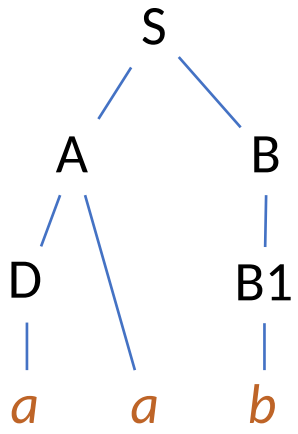


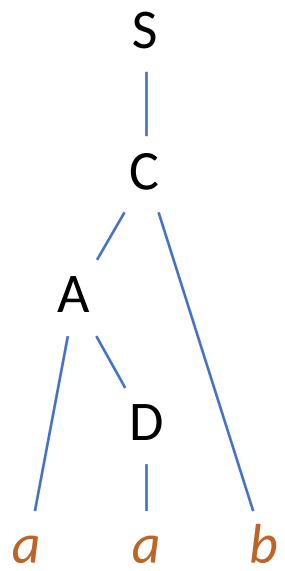


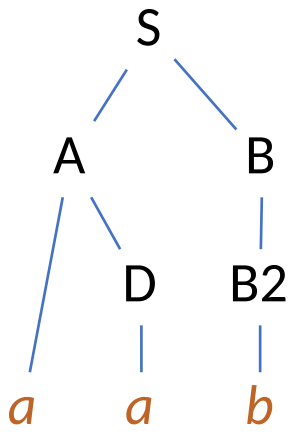
PCFG on an example

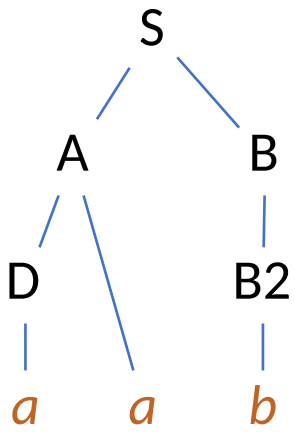
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

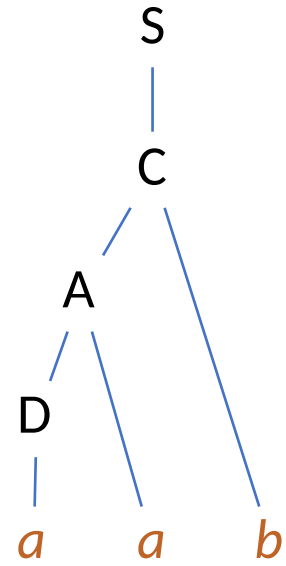






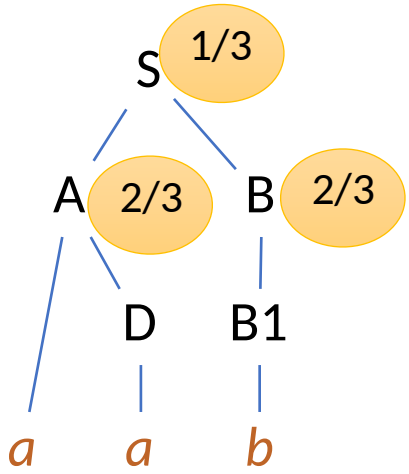


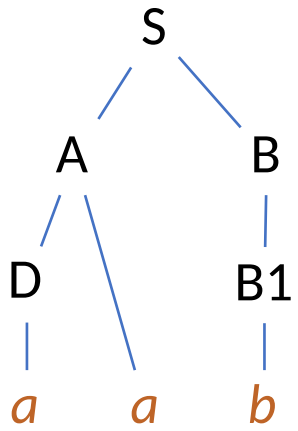


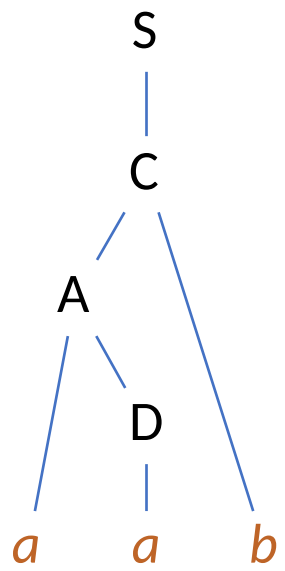


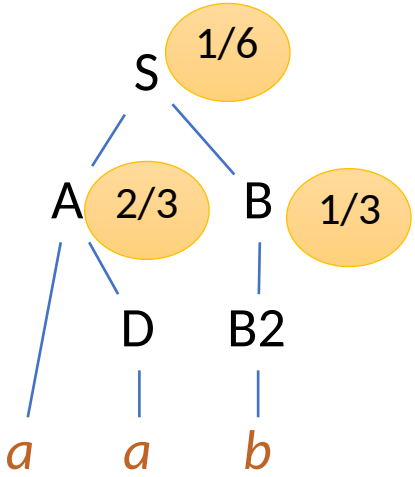
PCFG on an example

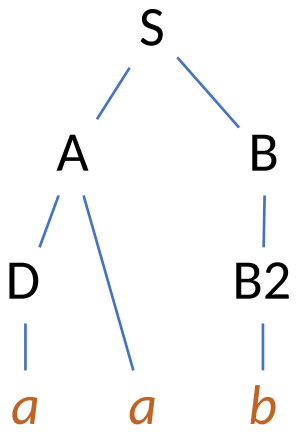
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

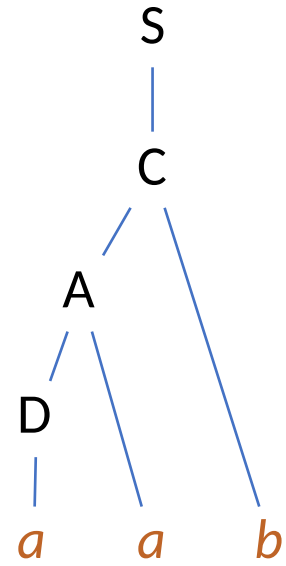






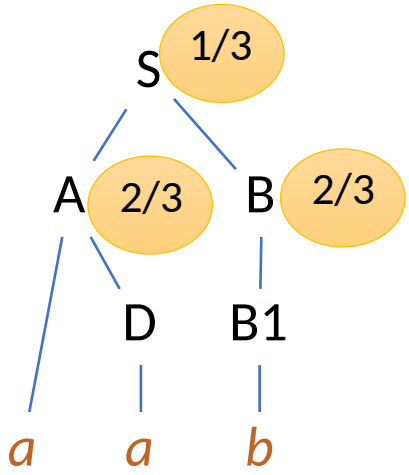


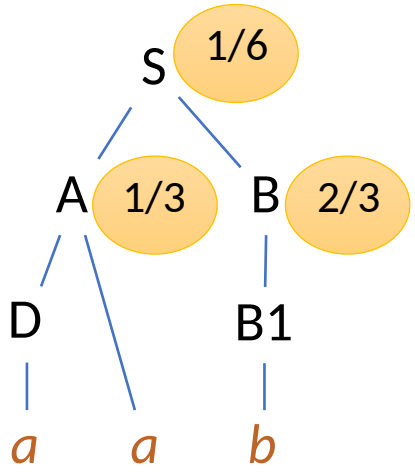


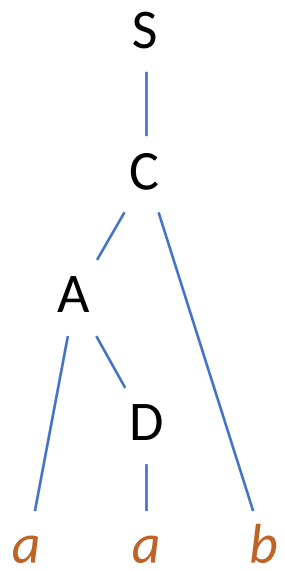


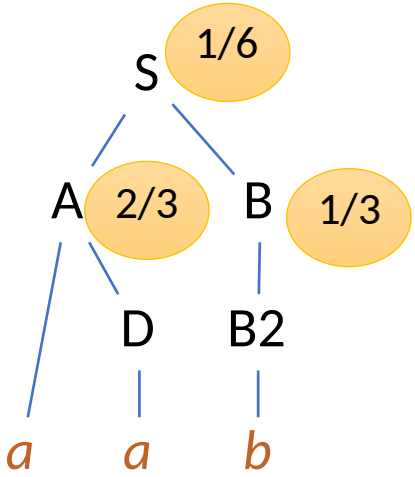
PCFG on an example

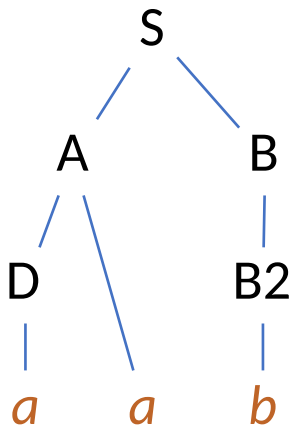
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

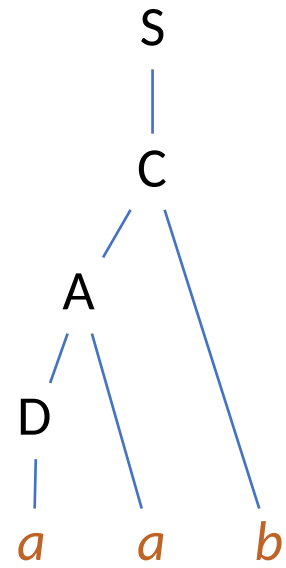






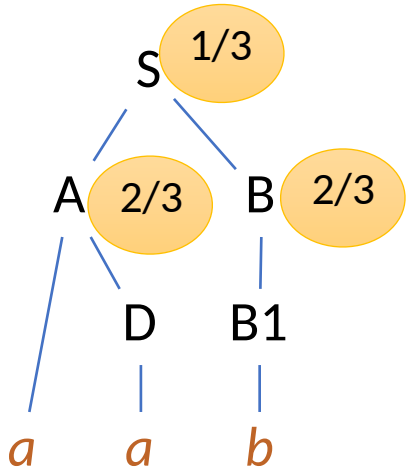


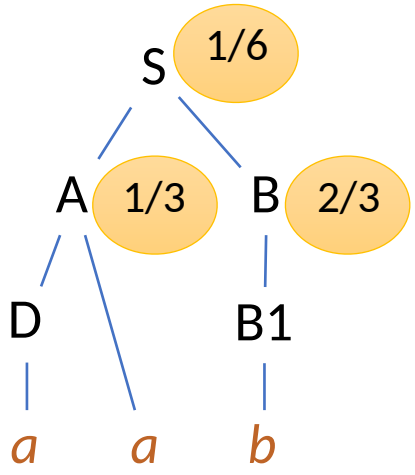


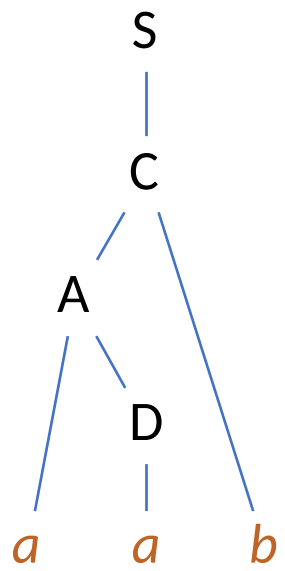


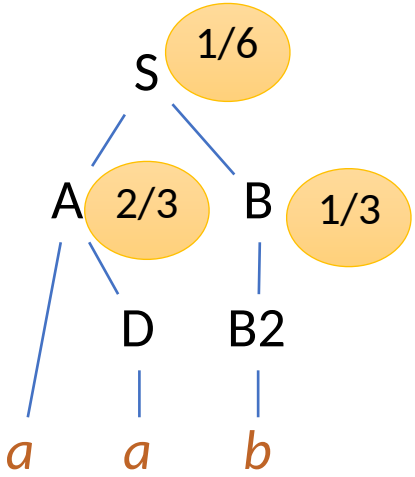
PCFG on an example

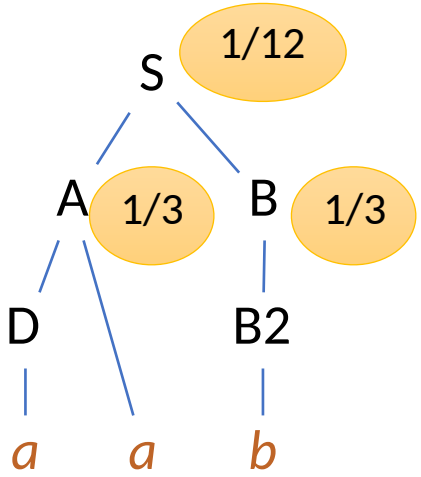
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

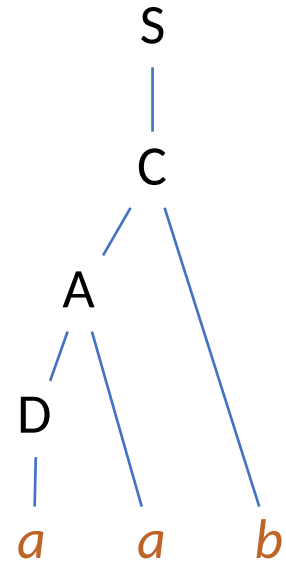






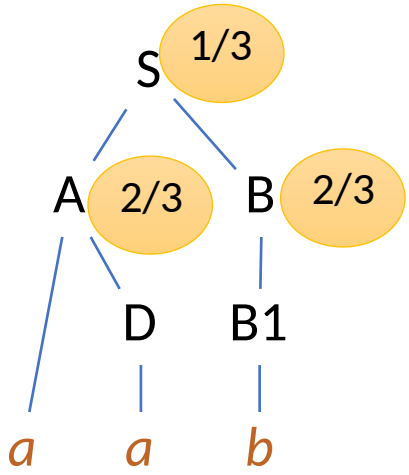


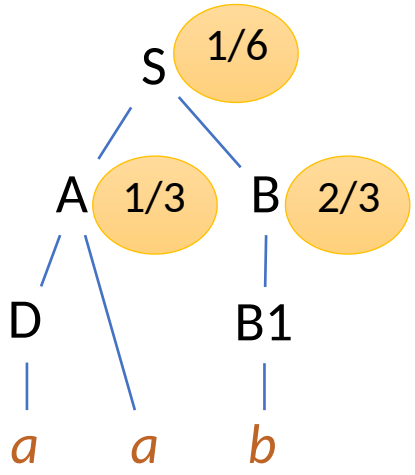


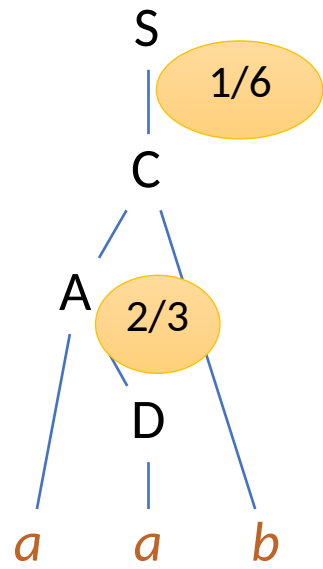


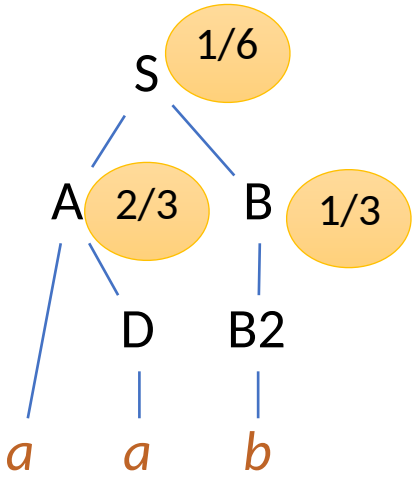
PCFG on an example

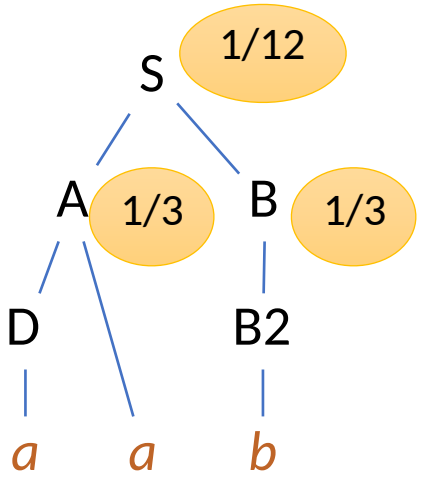
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

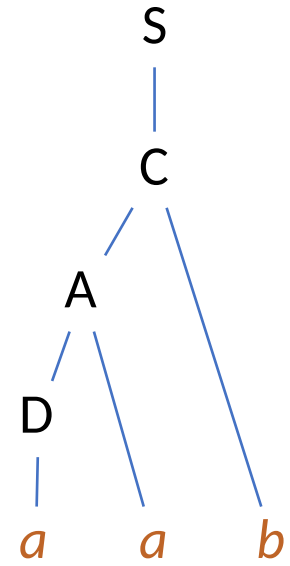






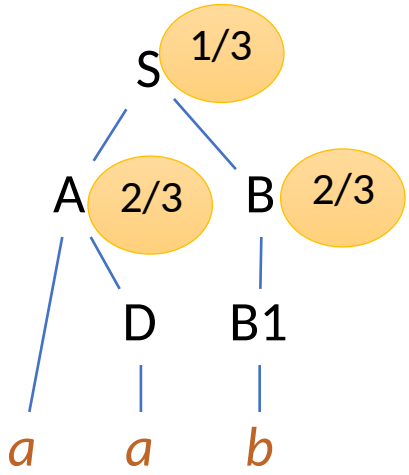


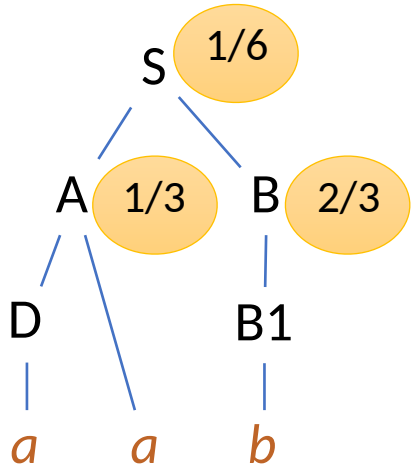


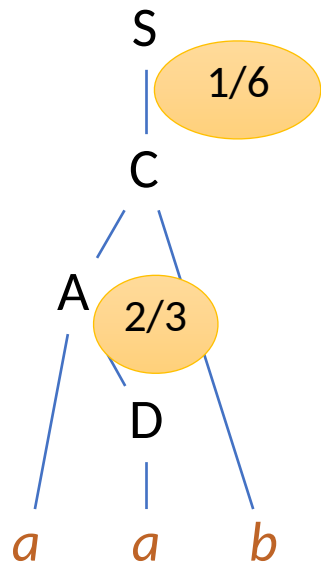


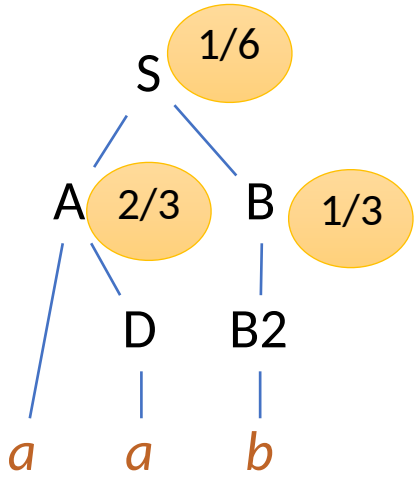
PCFG on an example

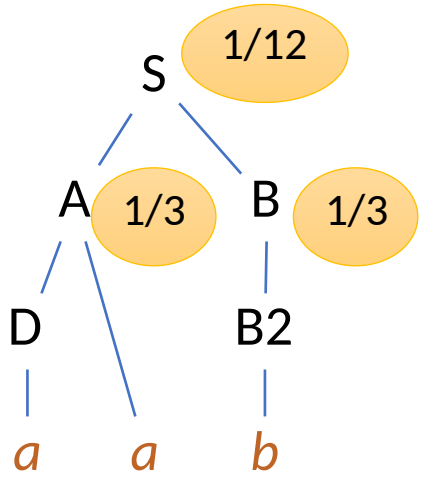
S -> A B	3/4	
S -> C		1/4
A -> a D	2/3	
A -> D a	1/3	
D -> a		1
C -> A b	1	
B -> B1		2/3
B -> B2		1/3
B1 -> b		1
B2 -> b		1

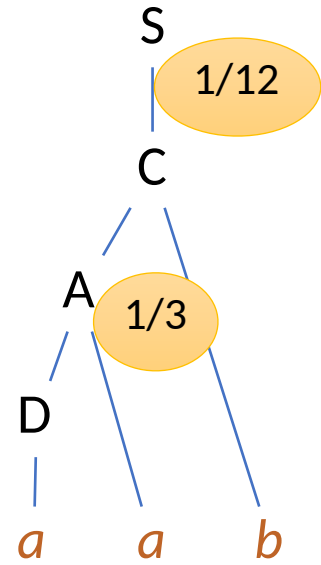




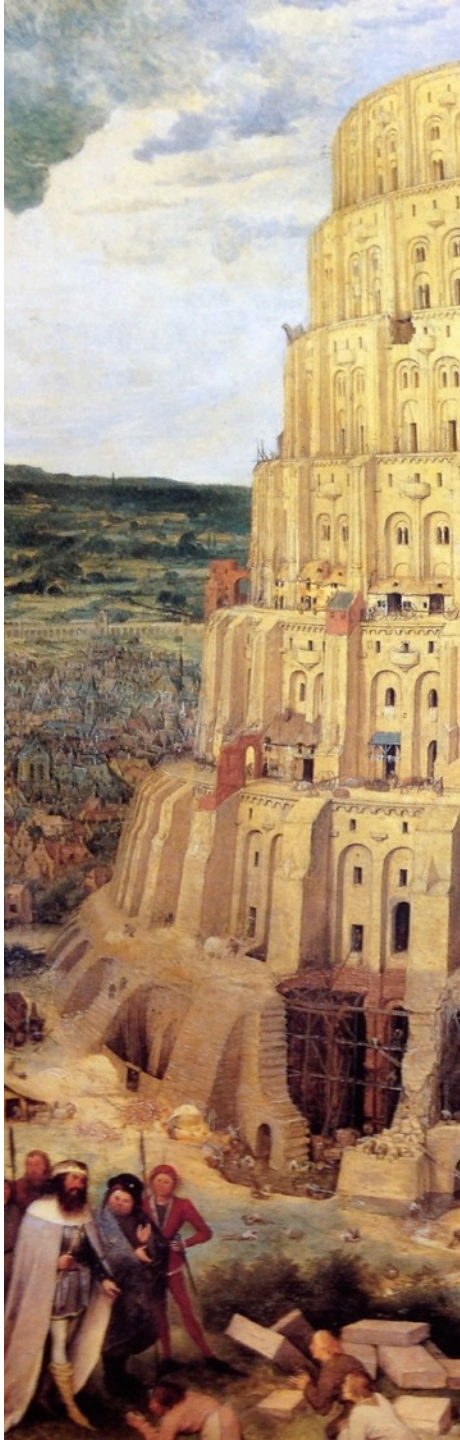






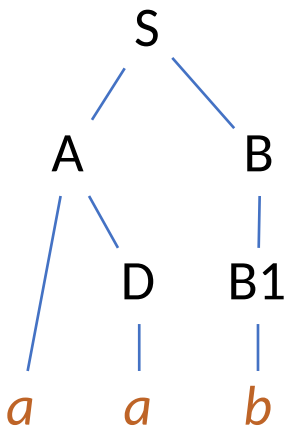


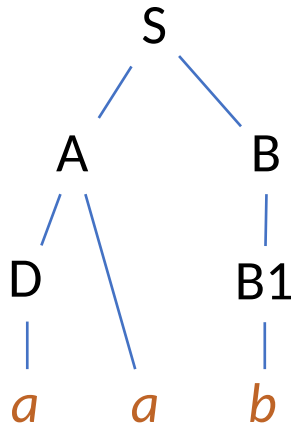
A few words
on shared parse forests

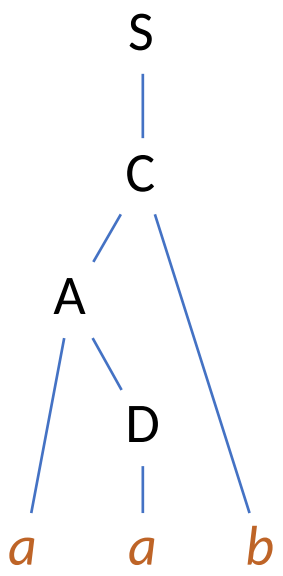


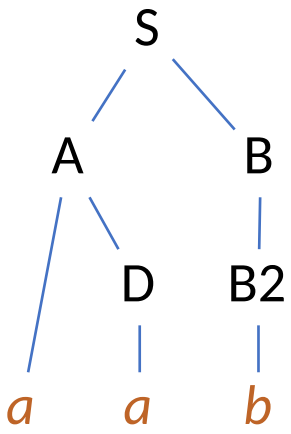
Multiple analyses: not very practical

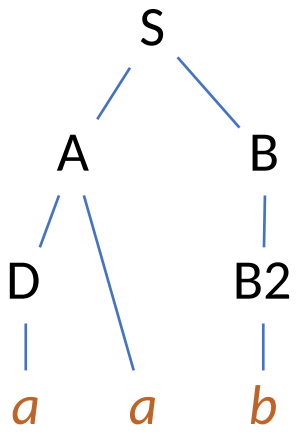
S -> A B
S -> C
A -> a D
A -> D a
D -> a
C -> A b
B -> B1
B -> B2
B1 -> b
B2 -> b

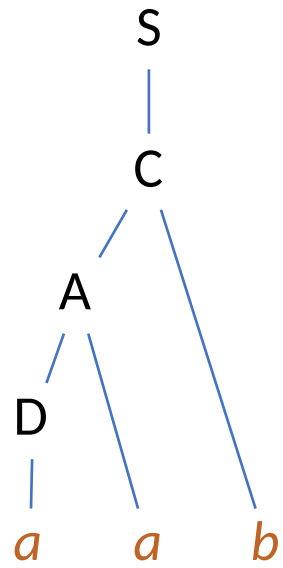






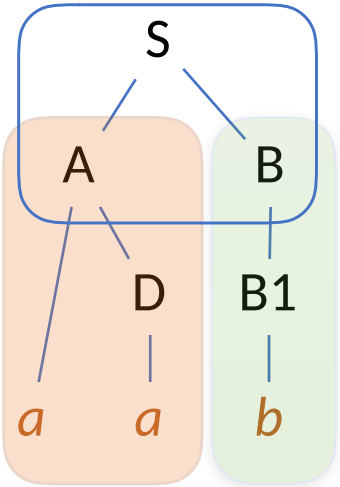


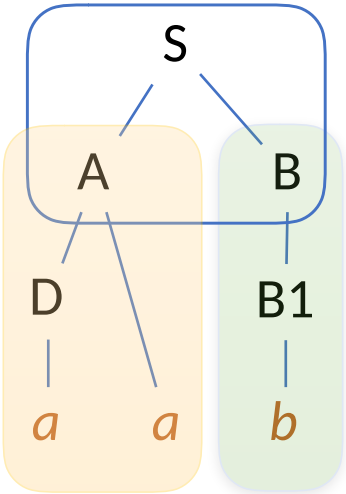


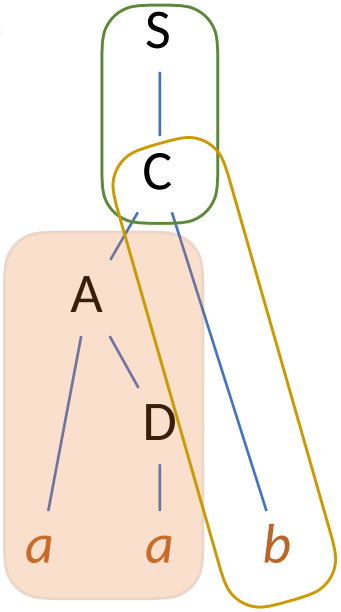


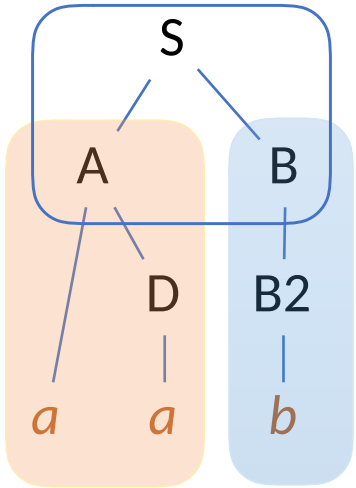
Multiple analyses: a lot of redundancies

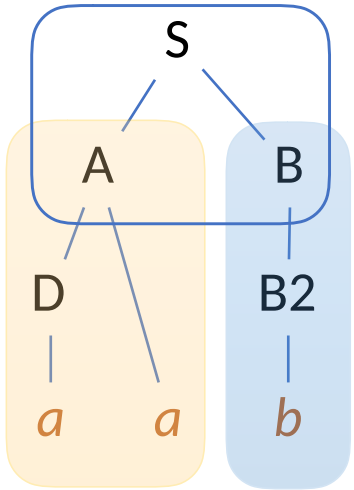
S -> A B
S -> C
A -> a D
A -> D a
D -> a
C -> A b
B -> B1
B -> B2
B1 -> b
B2 -> b

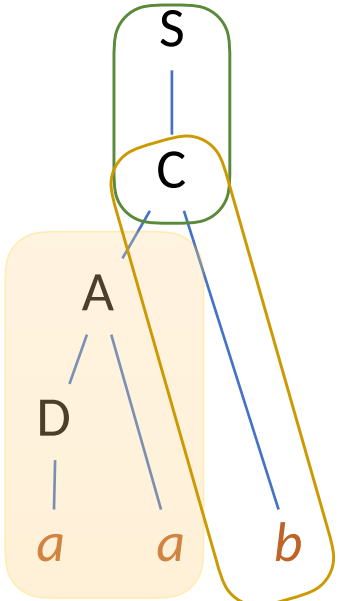






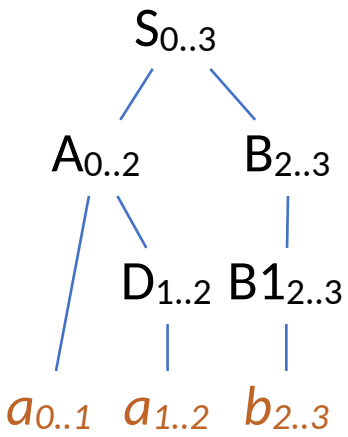


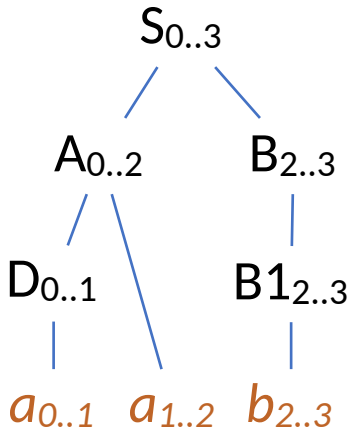


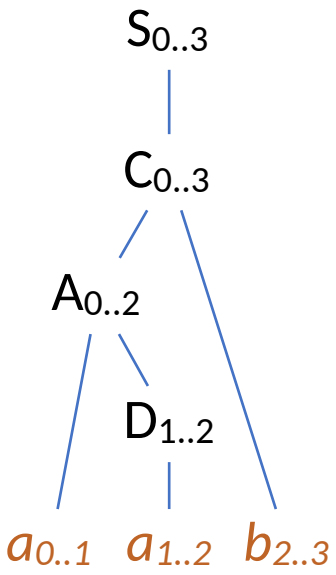


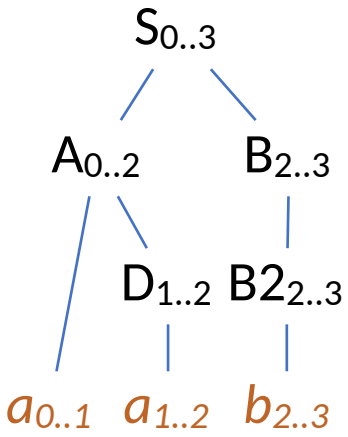
Multiple analyses: instanciation

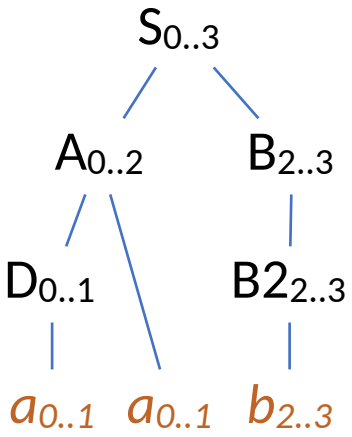
S -> A B
S -> C
A -> a D
A -> D a
D -> a
C -> A b
B -> B1
B -> B2
B1 -> b
B2 -> b

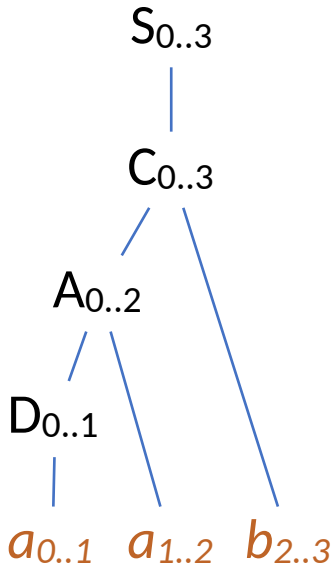






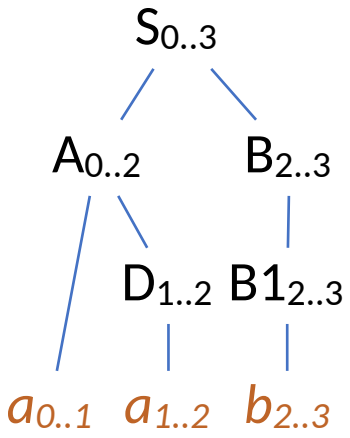






Towards parse forests

- Each parse can be represented as an **instanciated grammar**
  - Rules are instanciated rules of the original grammar
  - One original rule can be used more than once, with different instanciations



$S_{0..3}$ -> $A_{0..2}$ $B_{2..3}$

$A_{0..2}$ -> $a_{0..1}$ $D_{1..2}$

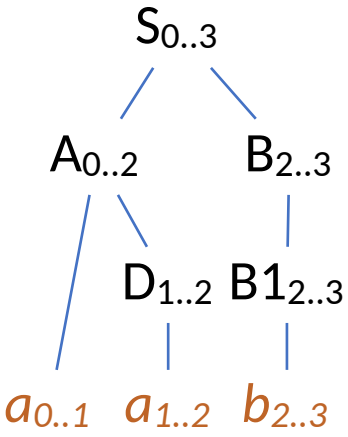
$D_{1..2}$ -> $a_{1..2}$

$B_{2..3}$ -> $B1_{2..3}$

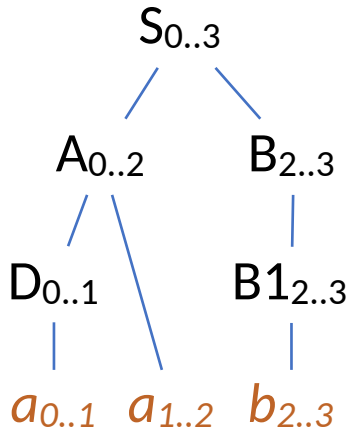
$B1_{2..3}$ -> $b_{2..3}$

Towards parse forests

- All grammars (one for each parse tree) have the same axiom, here $S_{0..3}$
- Whenever there are shared parts, they will appear in the form of repeated (redundant) rules



$S_{0..3}$ -> $A_{0..2}$ $B_{2..3}$
$A_{0..2}$ -> $a_{0..1}$ $D_{1..2}$
$D_{1..2}$ -> $a_{1..2}$
$B_{2..3}$ -> $B1_{2..3}$
$B1_{2..3}$ -> $b_{2..3}$



$S_{0..3}$ -> $A_{0..2}$ $B_{2..3}$
$A_{0..2}$ -> $D_{0..1}$ $a_{1..2}$
$D_{0..1}$ -> $a_{0..1}$
$B_{2..3}$ -> $B1_{2..3}$
$B1_{2..3}$ -> $b_{2..3}$

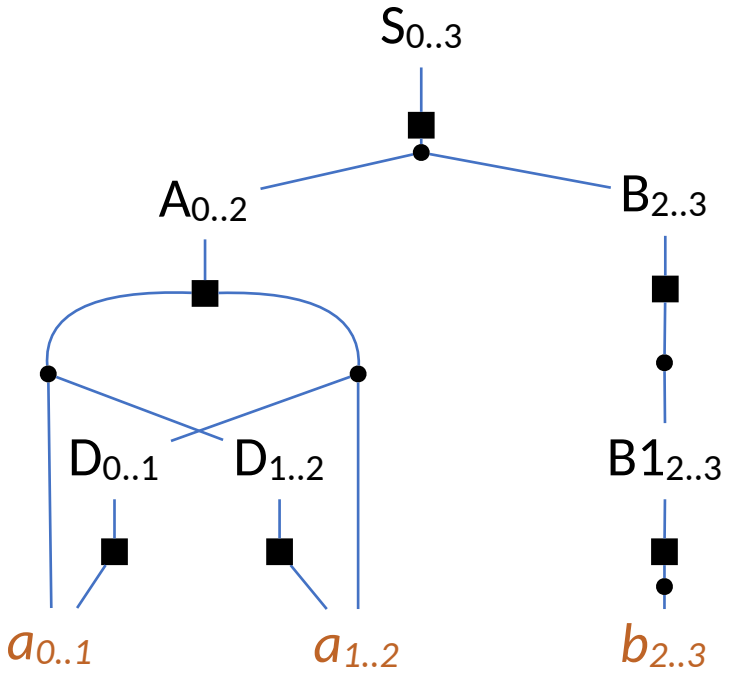
Towards parse forests

- All grammars (one for each parse tree) have the same axiom, here $S_{0..3}$
- Whenever there are shared parts, they will appear in the form of repeated (redundant) rules,
  - which can be merged!

$S_{0..3}$ -> $A_{0..2}$ $B_{2..3}$
$A_{0..2}$ -> $a_{0..1}$ $D_{1..2}$
$D_{1..2}$ -> $a_{1..2}$
$A_{0..2}$ -> $D_{0..1}$ $a_{1..2}$
$D_{0..1}$ -> $a_{0..1}$
$B_{2..3}$ -> $B1_{2..3}$
$B1_{2..3}$ -> $b_{2..3}$

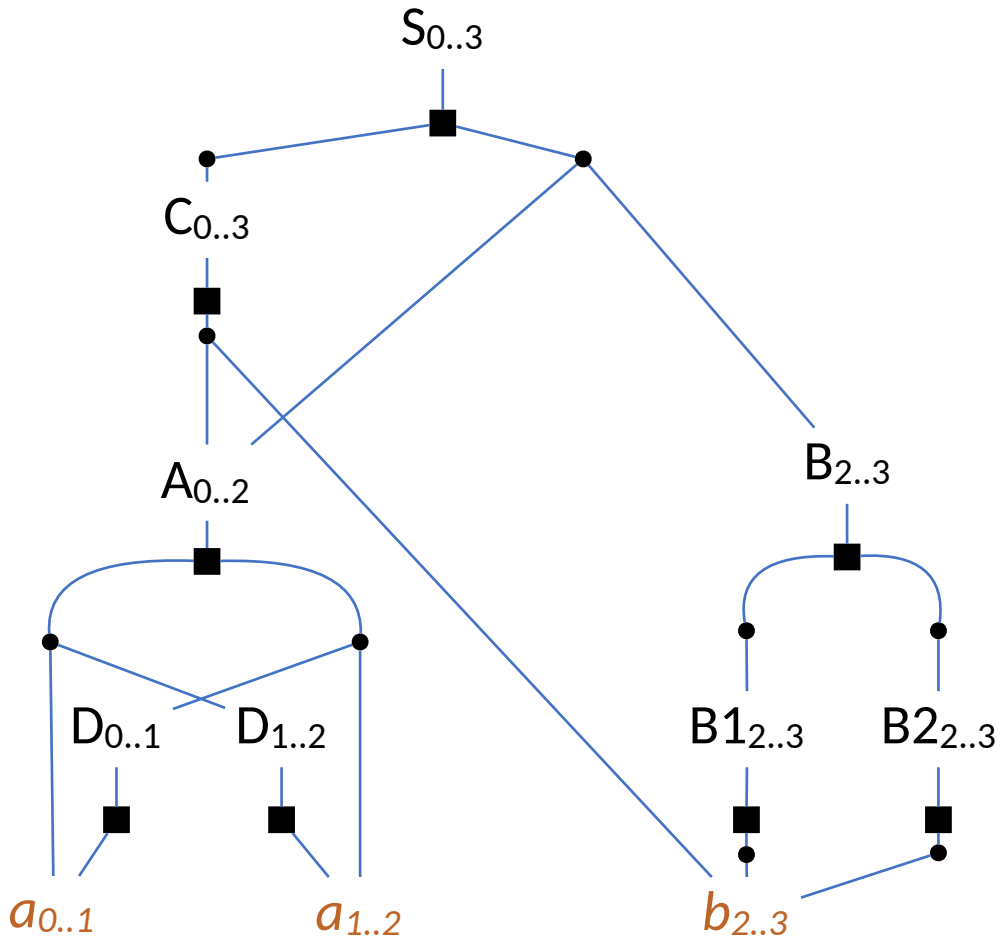
Towards parse forests

- This instanciated grammar defines a language containing the instanciated input string, and allows for two parses for this instanciated string
- We can represent this grammar as an AND-OR graph!



A (shared) parse forest

- We can represent all parses in this way



PCFGs and parse forests

- It is possible to extract the best (or the *n*-best) parse trees from a parse forest without extracting individual trees from the forest
- We do not have enough time for this today
  - For those interested, see Huang & Chiang (2005) “Better *k*-best parsing”
  - The algorithm takes advantage of the non-contextuality of both CFGs and PCFGs
- Parse forests are still a key data structure in parsing
  - But there are different approaches to the parsing problem, e.g. using discriminative algorithms
  - The role of neural networks is different: neural networks learn how to take good decisions; we still need a formal device to map a sequence of decisions into a parse tree


That’s all for today!