Computational Linguistics

CSC 2501 / 485
Fall 2015

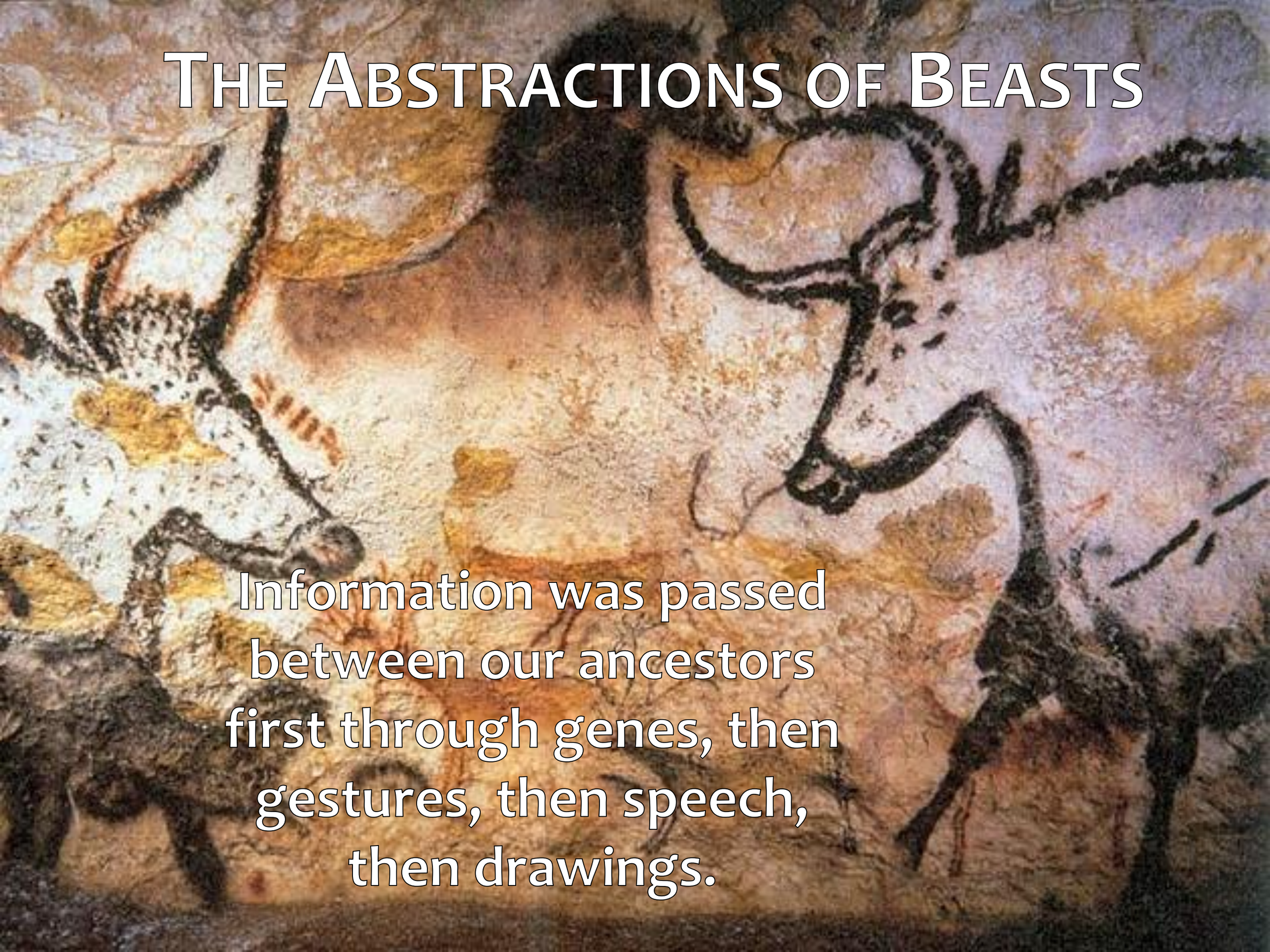
2

2. Introduction to syntax and parsing

Frank Rudzicz
Toronto Rehabilitation Institute-UHN; and
Department of Computer Science, University of Toronto

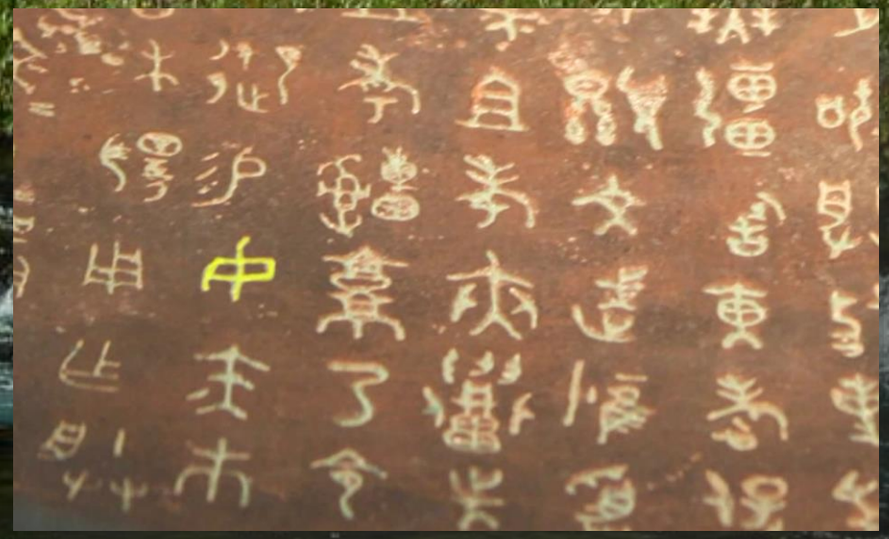
Reading: Jurafsky & Martin: 5.0–1, 12.0–12.3.3, 12.3.7, *[13.1–2]*.
Bird et al: 8.0–4.

The Abstractions of Beasts



Information was passed
between our ancestors
first through genes, then
gestures, then speech,
then drawings.

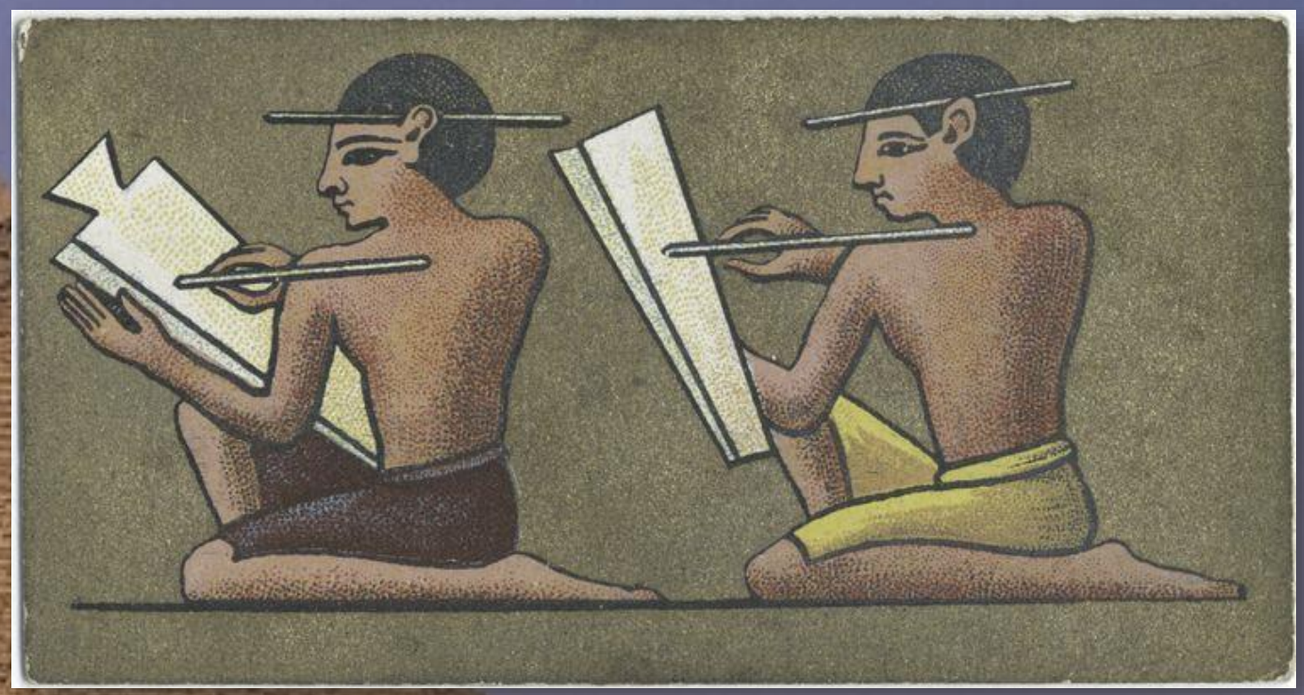
Imagine your ancestor wanted to leave the message "*there are ox halfway up the river*"



"OX" + "HALFWAY" + "RIVER" = message

IDEOGRAM

PICTOGRAMS



Ancient Egyptian (c. 3000 BCE)

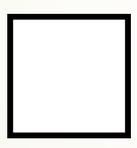
- **Few** writers
- **Stone** tablets
- Many (>1500) symbols representing ideas (e.g., *apple*)
- A few (~140) symbols representing sounds (e.g. *gah*)

- **Demotic (c. 650 BCE)**
  - **Many** writers
  - **Papyrus** sheets
- More **purposes** (e.g., recipes, contracts)
- Fewer symbols
- Higher **proportion** of symbols representing sounds

Writing systems

- **Logographic:** Symbols refer to **ideas.**
- **Phonographic:** Symbols refer to **sounds.**

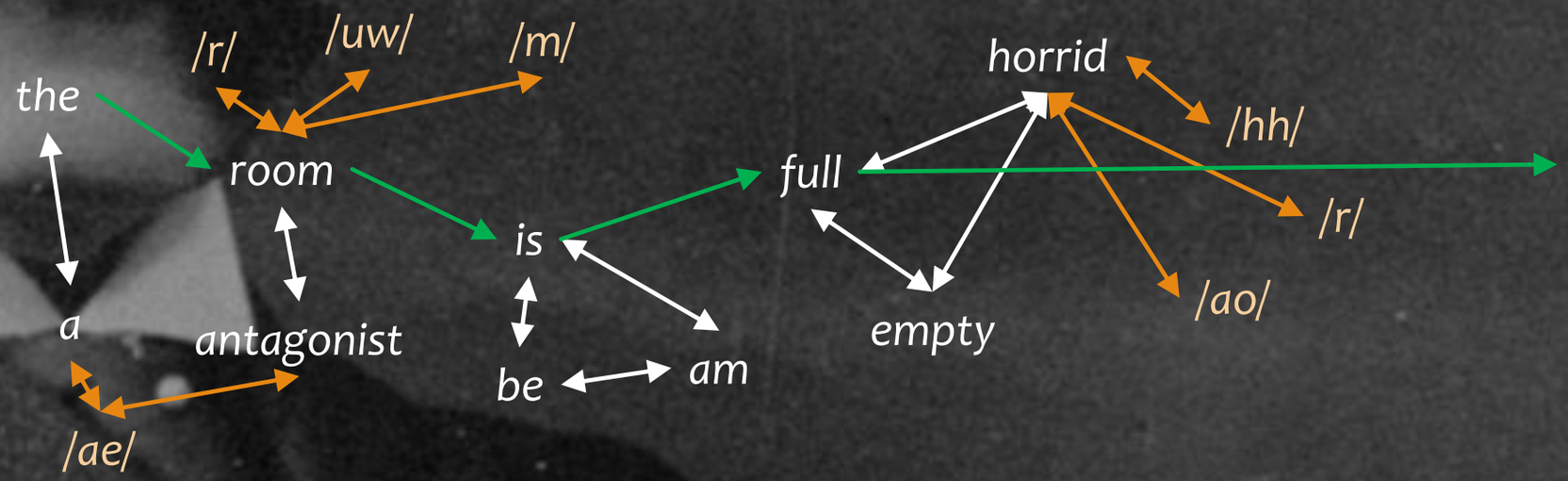
- English carries logographic heritage.

	“alph” (ox)	“bet” (house)	“kaf” (palm)	“mem” (water)	“en” (eye)	“ro” (head)
Proto-Sinaitic						
Phoenician						
Cyrillic	A	b	K	M	O	P

Structural linguistics

- Ferdinand de Saussure ('father of modern linguistics')
- 'Contrasts' and 'equivalents'
  - Sequences of words or phonemes.

    the room is full of milkmen, some of whom are very old.

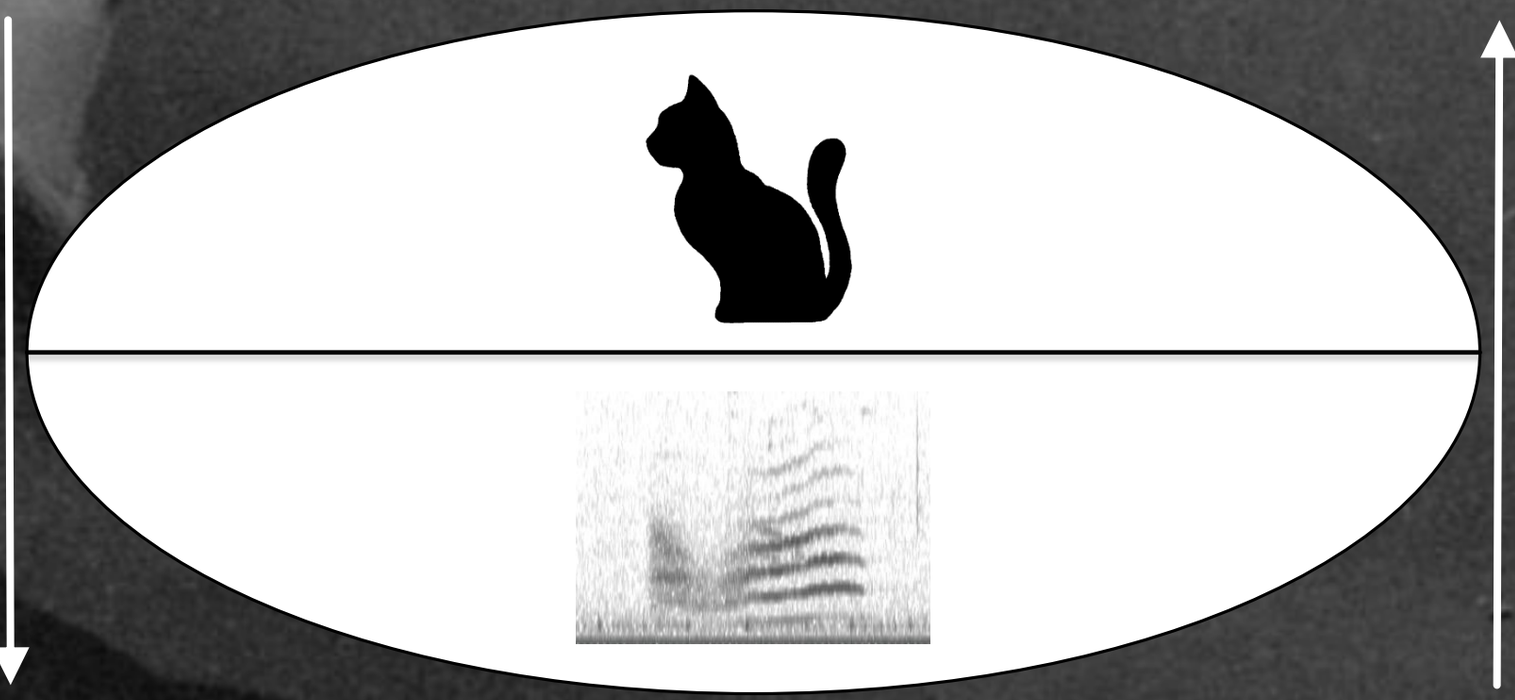


- These are defined only through contrasts with other objects

Saussure's 'sign'

signified

signifier



Structural linguistics

- The relationship between signifier and signified is arbitrary.
  - E.g., why should we call it '*cat*' and not '*chat*'? Convention. Society.
- Distinguish 'parole' (individual use) from 'langue' (system).
- synchrony: *n.* a snapshot of language use in time and place vs. diachrony: *n.* how a language evolves over time.
- syntagm: *n.* how words are aligned in a sequence linear sequence.

  You can't swap any word in this sentence.

  You can't this word any in swap sentence.

paradigm: *n.* how words *can* be replaced *some* of the time.

I like turtles. ✓ *The turtle walked.* ✓

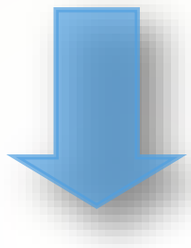
I like salmon. ✓ *The salmon walked.* X

Syntactic structure 1

- ***Syntax:***
  - The combinatorial structure of words.
  - How words can be hierarchically organized into ***phrases*** (e.g., [*that weasel*], [*snagged the bee*]), and ***sentences*** (e.g., [*that weasel snagged the bee*]).

Syntactic structure 2

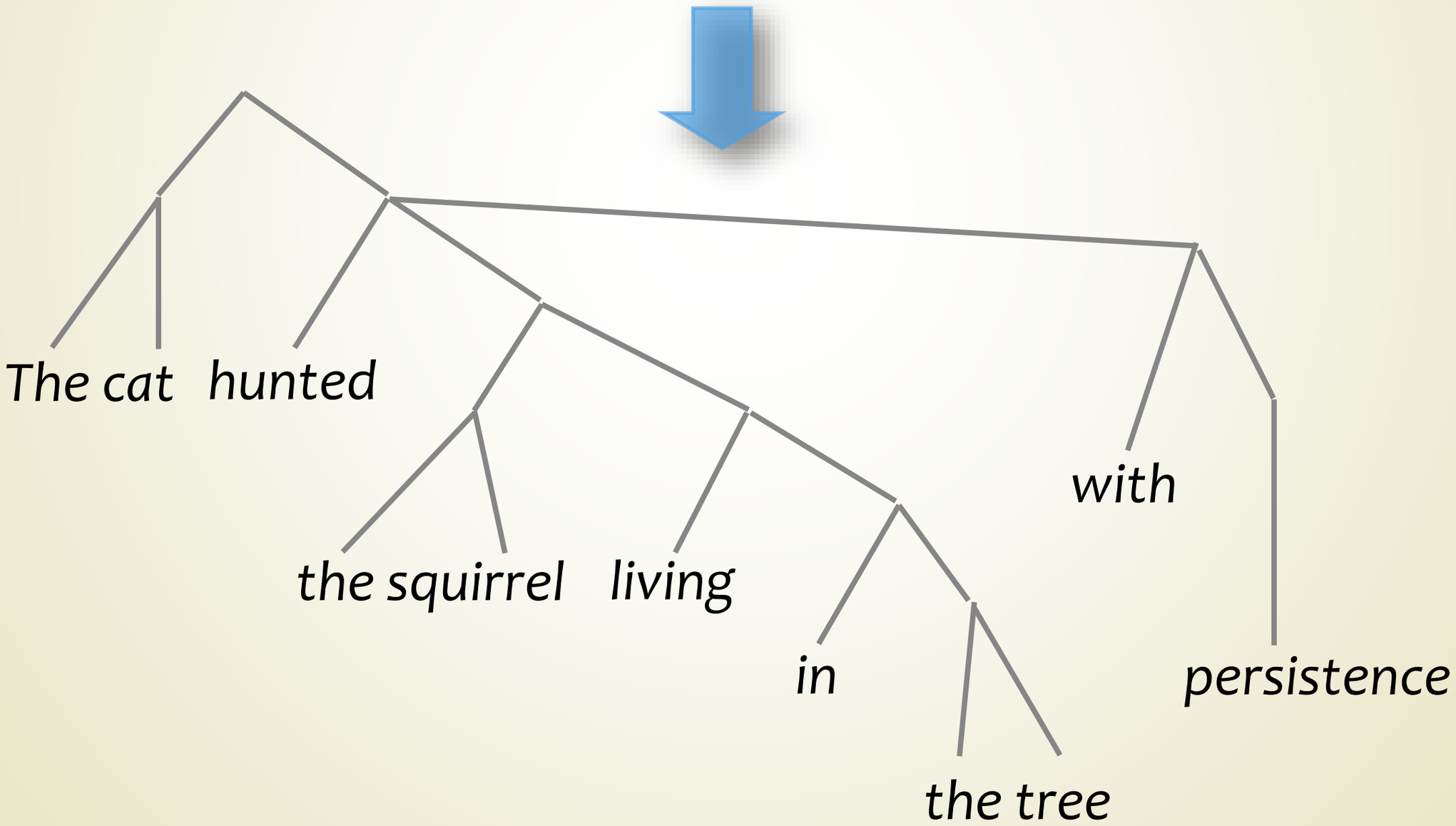
The cat hunted the squirrel living in the tree with persistence.



[[The cat]
[hunted [the squirrel [living [in [the tree]]]]
[with [persistence]]]]

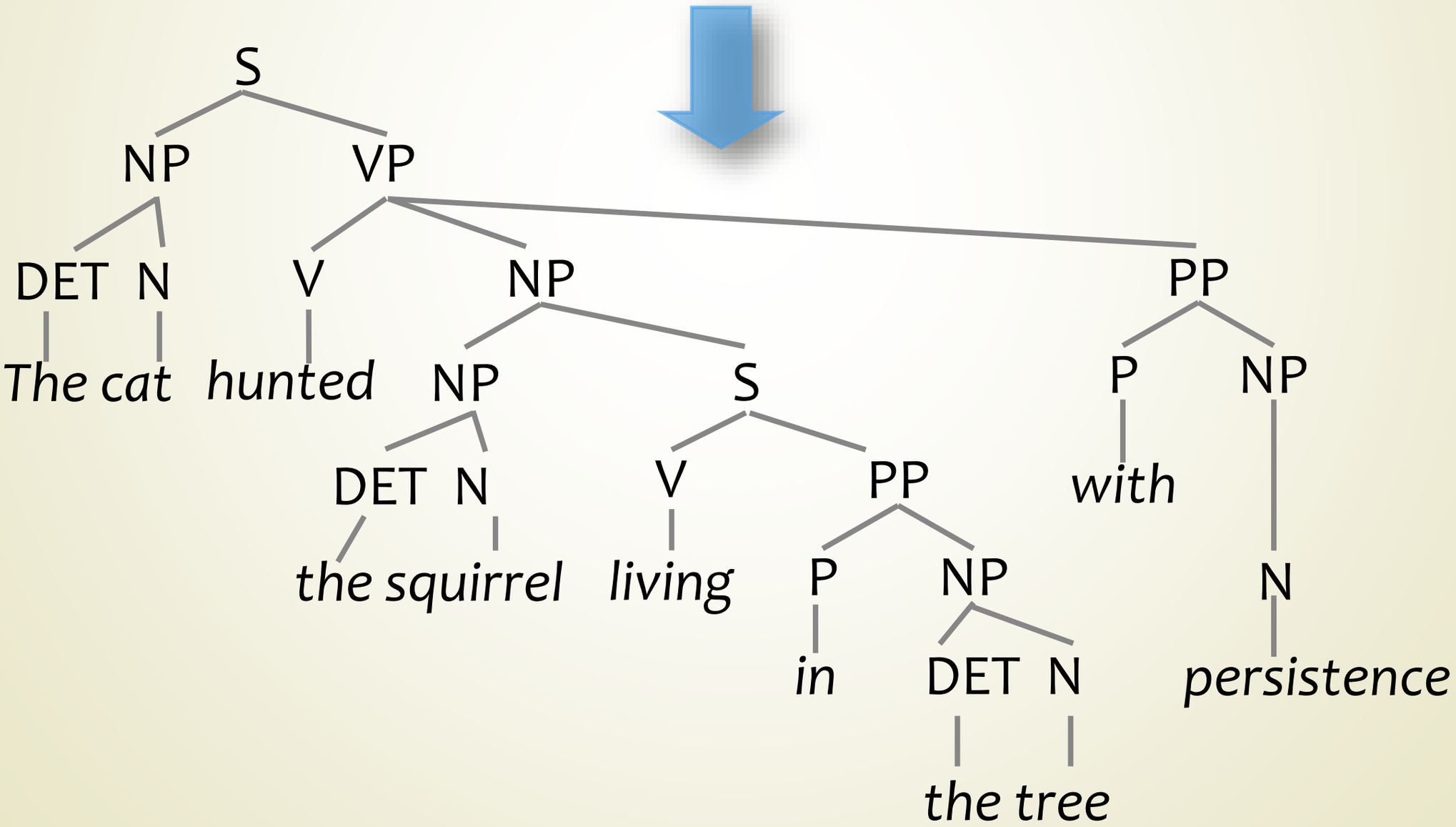
Syntactic structure 2

The cat hunted the squirrel living in the tree with persistence.



Syntactic structure 2

The cat hunted the squirrel living in the tree with persistence.



Syntactic structure 3

- ***Goal***: meaning, interpretation, semantics.
- So why do we care about syntax?

Grammars and parsing

- ***Grammar:***
  - **Formal specification** of allowable structures.
    - Knowledge
    - Representation
- ***Parsing:***
  - **Analysis** of string of words to determine the structure assigned by grammar.
    - Algorithm
    - Process

Using grammar to capture structure

- Main issues:
  - Which words are grouped together into a phrase.
  - How words within a phrase relate to a common **theme** (the **head** of the phrase).
  - How different phrases are related to each other.

- Use grammar to encode meaningful relations.

Good and bad grammars

- Many possible grammars for any natural language.
  - Some are better than others.
- Desiderata (*n.pl. things that are desired*):
  - Faithfulness to details of language.
  - Economy of description.
  - Reflects linguistic intuition.
  - Efficiency of parsing.

Elements of grammar

- **Primitives**: lexical categories or parts of speech.
  - Each ***word-type*** is a member of one or more.
  - Each ***word-token*** is an instance of exactly one.
- Categories are ***open*** or ***closed*** to new words.
- Eight main categories, many subcategories.

Nine Seven

Twenty-three

Parts of speech 1

- ***Nouns:*** denote an object, a concept, a place, …
  - **Count nouns:** *dog, spleen, Band-Aid, …*
  - **Mass nouns:** *water, wheat, …*
  - **Proper nouns:** *Shanaenae, Toronto, …*
- ***Pronouns:*** *he, she, you, I, they, …*
- ***Adjectives:*** denote an attribute of the denotation of a noun.
  - **Extensional:** *pink, furry, …*
  - **Measure:** *big, …*
  - **Intensional:** *former, alleged, …*

Parts of speech 2

- ***Determiners, articles:*** specify certain attributes of the denotation of a noun that are grammatically relevant.
  - *the, a, some, …*
- ***Verbs:*** predicates, denote an action or a state.
  - **Intransitive:** *sleep, die, …*
  - **Transitive:** *eat, kiss, …*
  - **Bi-transitive:** *give, sell, …*
  - **Copula:** *be, feel, become, …*

Parts of speech 3

- ***Adverbs:*** denote an attribute of the denotation of a predicate.
  - **Time and place:** *today, there, now, …*
  - **Manner:** *happily, furtively, …*
- ***Prepositions:*** relate two phrases with a location, direction, manner, etc.
  - *up, at, with, in front of, before, …*
    - X *"this is the kind of B.S. I won't put* ***up with****"*
    - ✓ *"this is the kind of B.S.* ***up with*** *which I will not put"*

Parts of speech 4

- ***Conjunctions:*** combine two clauses or phrases:
  - **Coordinating conjunctions:** *and, or*
    - *"the sound **and** the fury"*
  - **Subordinating conjunctions:** *but, while, …*
- ***Interjections:*** stand-alone exclamations.
  - *um, wow, oh dear, balderdash, crikey, …*

Elements of grammar

- Combinations:
  - **Phrase:** a hierarchical grouping of words and phrases.
  - **Clause:** a grouping that includes a verb phrase at its top level.
  - **Sentence:** a grouping of one or more clauses.
- Can be represented by tree or by labelled bracketing.
- Terminology: A ***constituent*** is any well-formed element (word, phrase, or clause).

Types of phrase 2

- Noun phrase (NP):
  - *a mouse*
  - *mice*
  - *the handsome marmot*
  - *the handsome marmot on the roof*
- Verb phrase (VP):
  - *Stepped lightly*
  - *quickly gave the Telefunken U47 to Mary*

Types of phrase 2

- Adjective phrase (AP):
  - *green*
  - *proud of Kyle*
  - *very happy that you went*

- Prepositional phrase (PP):
  - *in the sink*
  - *without feathers*
  - *astride the donkey*

Clauses and sentences 1

- Clauses:
  - *Ross remarked upon Nadia's dexterity*
  - *to become a millionaire by the age of 30*
  - *that her mother had lent her for the banquet*

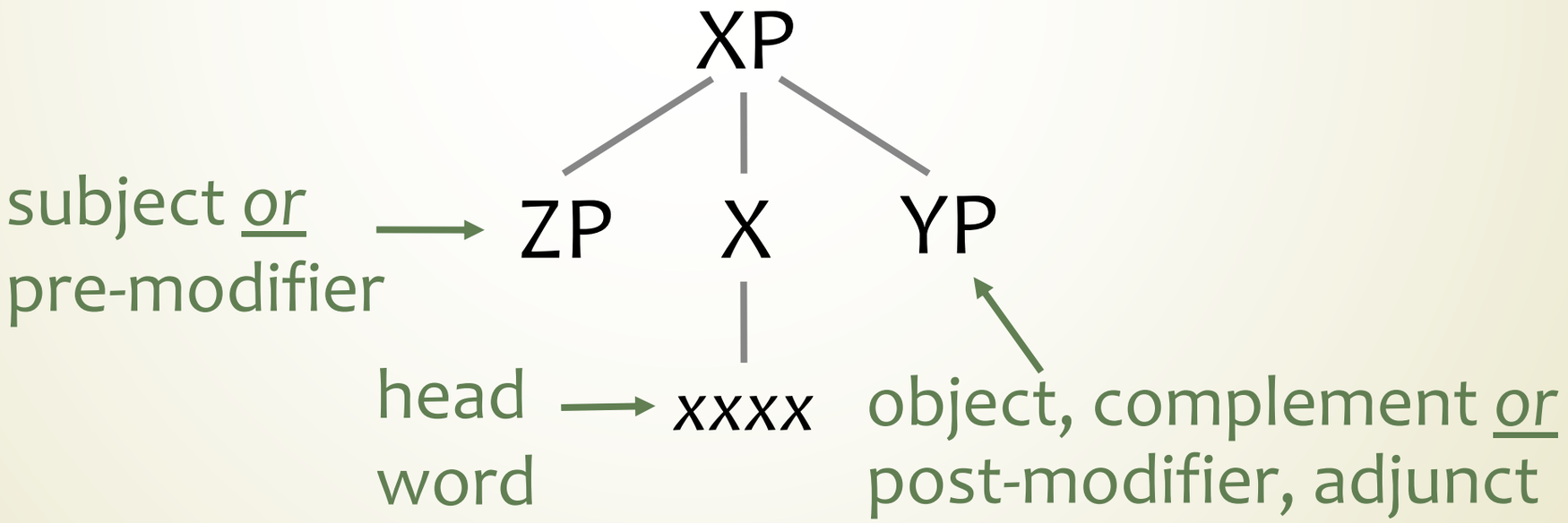
- Sentences:
  - *Ross remarked upon Nadia's dexterity.*
  - *Nathan wants to become a millionaire by the age of 30.*
  - *Nadia rode the donkey that her mother had lent her for the banquet.*
  - *The handsome marmot on the roof.*

Clauses and sentences 2

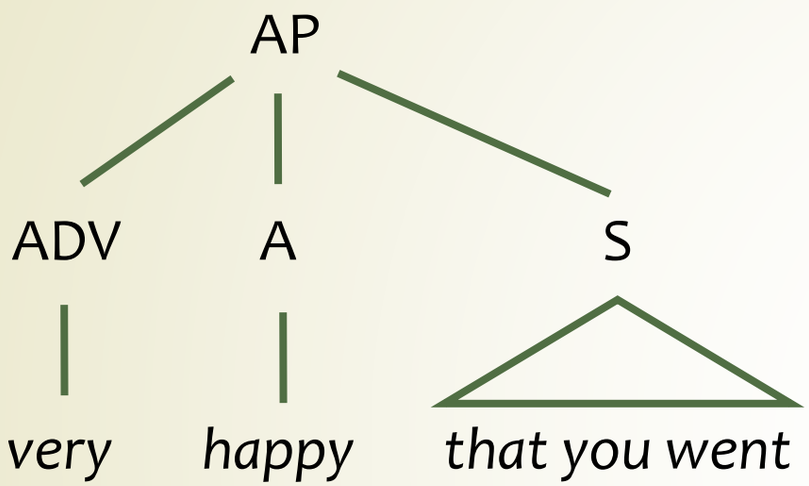
- Clauses may act as noun phrases:
  - ***To become a millionaire by the age of 30*** *is what Ross wants.*
  - ***Nadia riding her donkey*** *is a spectacular sight.*
  - *Ross discovered that* ***Nadia had been feeding his truffles to the donkey****.*

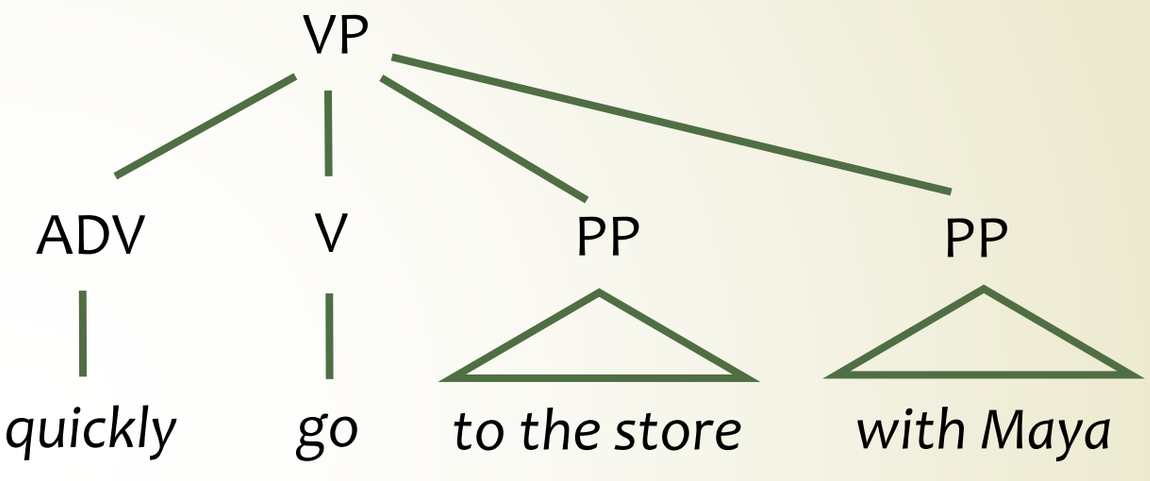
The structure of an idealized phrase

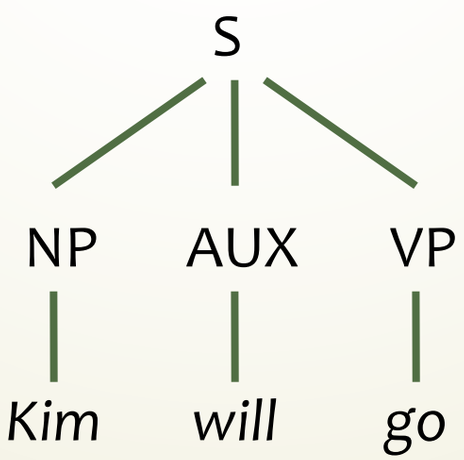
$XP \rightarrow ZP \;\; X \;\; YP$



Example phrases







Formal definition of a CFG

- A **context-free grammar (CFG)** is a quadruple $G = (V_t, V_n, P, S)$, where
  - V_t is a finite set of **terminal** symbols.
  - V_n is a finite set of **non-terminal** symbols.
  - P is a finite set of **production rules** of the form
    $$A \longrightarrow \alpha$$
    where $A \in V_n$ and α is a sequence of symbols in $(V_n \cup V_t)^*$.
  - $S \in V_n$ is the **start** symbol.

Terminology

- **Non-terminal** (NT):

  A symbol that occurs on the left-hand side (LHS) of some rule.

- **Terminal** (T):

  A symbol that never occurs on the LHS of a rule.

- **Start symbol:**

  A specially-designated NT that must be the *root* of any tree derived from the grammar.

  In our grammars, it is usually **S** for sentence.

A simple grammar

S = S, P = {

S → NP VP

NP → Det N

NP → Det Adj N

NP → NP PP

VP → V

VP → V NP

PP → P NP

Det → *the* | *a* | *an*

Adj → *old* | *red* | *happy* | …

N → *dog* | *park* | *statue* | *contumely* | *run* | …

V → *saw* | *ate* | *run* | *disdained* | …

P → *in* | *to* | *on* | *under* | *with* | … }

V_t and V_n can be inferred from the production rules.

The lexicon:
In practice, a separate data structure

Lexical categories:
NT's that rewrite as a single T.

Parsing 1

- ***Parsing:*** Determining the structure of a sequence of words, given a grammar.
  - Which grammar rules should be used?
  - To which symbols (words / terminals and nodes / non-terminals) should each rule apply?

Parsing 2

- Input:
  - A context-free grammar.
  - A sequence of words

    Time flies like an arrow

    or, more precisely, of sets of parts of speech.

    {noun,verb} {noun,verb} {verb,prep} {det} {noun}
- Process:
  - Working from left to right, ***guess*** how each word fits in.

Parsing 3

- If a guess leads to failure (parse is stymied), ***back up to a choice point*** and try a different guess.
  - Backtracking, non-determinism.
  - At each guess, must save state of parse on a stack.
  - (Or, explore in parallel.)
- Want to guess right:
  - Order of preference for rules.

Parsing 4

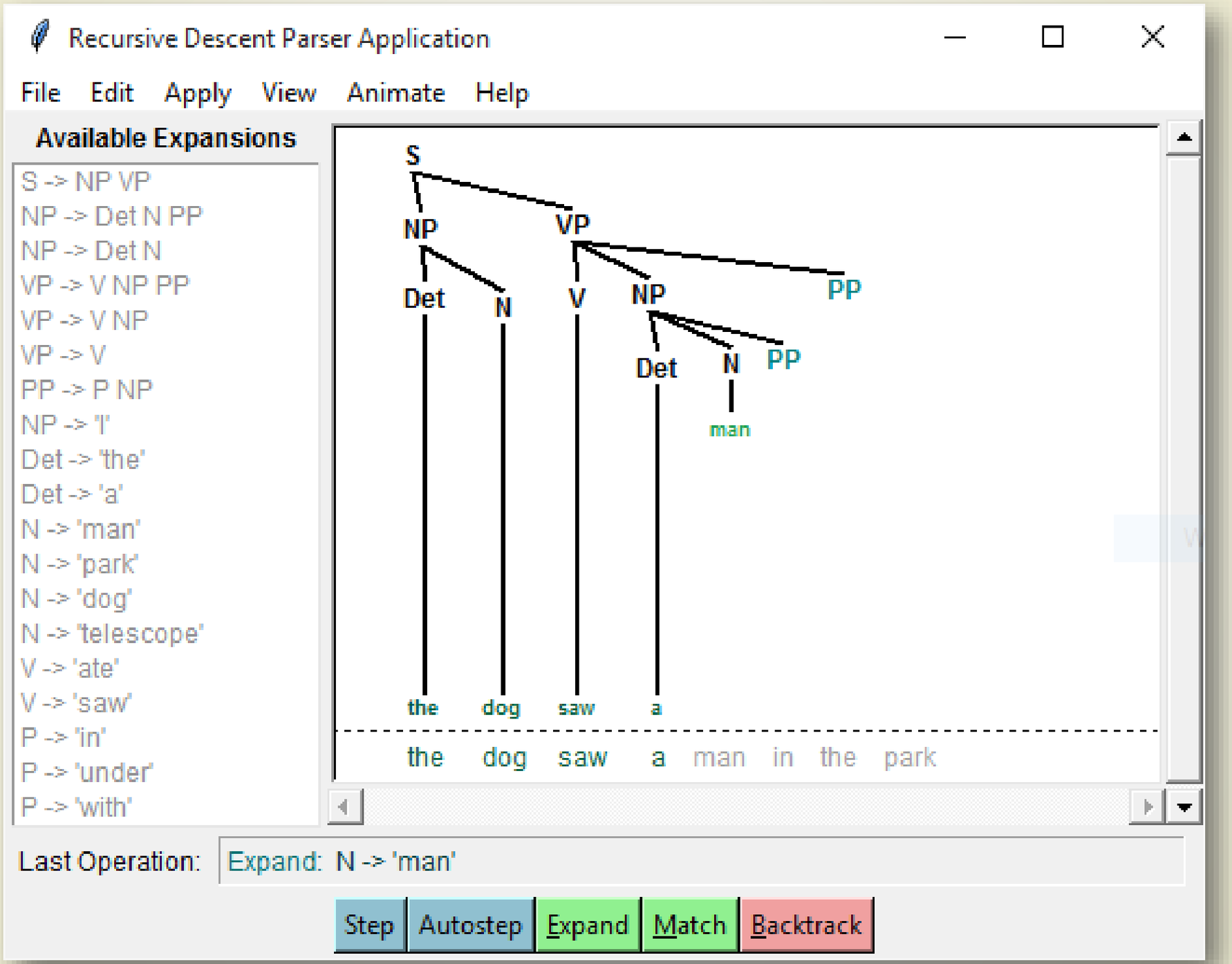
- Parsing can be formulated as a search problem.
  - Top-down.
  - Bottom-up.

Top-down parsing 1

- ***Top-down*** or ***rule-directed*** parsing:
  "*Can I take these rules and match them to this input?*"
  - Initial goal is an S.
  - Repeatedly look for rules that decompose /expand current goals and give new goals.
    E.g., goal of S may decompose to goals NP and VP.
  - Eventually get to goals that look at input.
    E.g., goal of NP may decompose to *det* or *noun.*
  - Succeed iff entire input stream is accounted for as S.

Top-down parsing 2

- Example: A ***recursive descent parser***.
  `>>> nltk.app.rdparser()`
- Operations on ***leftmost frontier node***:
  - ***Expand*** it.
  - ***Match*** it to the next input word.


Recursive Descent Parser Application
File Edit Apply View Animate Help
Available Expansions
S -> NP VP
NP -> Det N PP
NP -> Det N
VP -> V NP PP
VP -> V NP
VP -> V
PP -> P NP
NP -> 'I'
Det -> 'the'
Det -> 'a'
N -> 'man'
N -> 'park'
N -> 'dog'
N -> 'telescope'
V -> 'ate'
V -> 'saw'
P -> 'in'
P -> 'under'
P -> 'with'
S
NP
VP
Det
N
V
NP
PP
Det
N
PP
man
the
dog
saw
a
the dog saw a man in the park
Last Operation:
Expand: N -> 'man'
Step
Autostep
Expand
Match
Backtrack

Top-down parsing 3

- Choice of next operation (in NLTK demo):
  - If it's a terminal, try matching it to input.
  - If it's a non-terminal, try expanding with first-listed untried rule for that non-terminal.

Bottom-up parsing 1

- ***Bottom-up*** or ***data-directed*** parsing:
  "Can I take this input and match it to these rules?"
  - Try to find rules that match a possible PoS of the input words …
  - … and then rules that match the constituents thus formed.
  - Succeed iff the entire input is eventually matched to an S.

Bottom-up parsing 2

- Example: A ***shift–reduce parser***.
  ```
  >>> nltk.app.srparser()
  ```
- Operations:
  - ***Shift*** next input word onto stack.
  - Match the top *n* elements of stack to RHS of rule, ***reduce*** them to LHS.

Shift Reduce Parser Application

File Edit Apply View Animate Help

Available Reductions

- S -> NP VP
- NP -> Det N
- NP -> NP PP
- VP -> VP PP
- VP -> V NP PP
- VP -> V NP
- PP -> P NP
- NP -> 'I'
- Det -> 'the'
- Det -> 'a'
- N -> 'man'
- V -> 'saw'
- P -> 'in'
- P -> 'with'
- N -> 'park'
- N -> 'dog'
- N -> 'statue'
- Det -> 'my'

Stack	Remaining Text
NP V NP Det N saw Det N my dog a man	in the park with a statue

Last Operation: Reduce: NP -> Det N

Step Shift Reduce Undo

Bottom-up parsing 3

- Choice of next operation (in NLTK demo):
  - Always prefer reduction to shifting.
  - Choose the first-listed reduction that applies.
- Choice of next operation (in real life):
  - Always prefer reduction to shifting for words, but not necessarily for larger constituents.

Problems

- Neither top-down nor bottom-up search exploits properties of CFG rules.
- **Problems:**
  - Recomputation of constituents.
  - Recomputation of common prefixes.
- **Solution:** Keep track of:
  - Completed constituents.
  - Partial matches of rules.