SYNTAX

Matt Post

IntroHLT class

~~10~~ 14 September 2026

JOHNS HOPKINS
UNIVERSITY

Languages

	GOOD	BAD
python program	for i in range(args.N): print(i)	i in ?: j for range(print)

JSON	{
	  “name": "Pierre",
	  "age": 61,
	  "tags": ["board", "director"]
	}
		"age": {
		  61,
		  "Pierre" ["board": “director"]
		  ‘name" "tags"
		}

URLs	http://google.com	gsd@ht//:ww
language	The crowd could not keep back gasps of admiration	not of could back gasps The crowd admiration

What are the abstractions and tools that explain these good and bad examples?

This is a question of ***syntax***

Today we will cover

math	**linguistics**	**engineering**
formal language theory	natural language	parsing
abstractions for reasoning about structure	*applying structure to natural phenomena*	*making them usable by a computer*

Goals for today

- After today, you should be able to
  - **define** a language
  - **describe syntax** both mathematically and linguistically
  - **enumerate** the formal language (Chomsky) hierarchy
  - **provide a description** of constituent grammars
  - **sketch the algorithm** for CKY parsing

Outline

formal language theory

natural language

parsing

Formal Language Theory

- A **language** is a set of strings under some alphabet, Σ
  - Σ = a set of symbols (letters, words, numbers, etc)
  - string = a sequence of symbols from Σ
- e.g., the set of valid English sentences (where the “alphabet” is English words), or the set of valid Python programs

Some notes

- Σ^* ("sigma star") is the set of all strings in the vocabulary
- ϵ is the *empty string*
- A language $\mathscr{L}$ can be very large—even infinite!

Languages as sets

- $\Sigma = \{0,1,2,3,4,5,6,7,8,9,.,-\}$

- What do you think these languages describe (in words?)

$$\mathscr{L}_1 = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ \ \ldots\}$$

$$\mathscr{L}_2 = \{-12.4, 0,\ 142,\ 142.1,\ 142.01,\ 142.001,\ \ \ldots\}$$

Formal Language Theory

- Formal Language Theory provides a common framework for studying properties of these languages, e.g.,
  - Is this file a valid C++ program? A valid Czech sentence?
  - What is the structure? <=> How do I find its meaning?
  - How hard / time-consuming is it to answer these questions?

Generative descriptions of lang.

- A definition of languages as **sets** is not very useful
  - Why not?
- A better approach:
  - Develop a process that can describe how strings can be determined to be in a language
  - New membership criteria:
    - **IN**: can be generated by this process
    - **OUT**: cannot be generated by this process
-

Regular expression examples

- $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
- A more compact representation: **regular expressions**
- What do these languages describe (in words?)

  $\mathscr{L}_1 = \Sigma^*$

  $\mathscr{L}_2 = 0 \mid [1-9][0-9]*$

- Notes:
  - | = “or”
  - [] = “choose one of these”
  - + = “one or more of the previous”
  - * = “zero or more of the previous”

Generating from a language

- Imagine a process that repeatedly
  - begins by choosing a symbol
  - continues to do so until it decides to terminate
- How can we formalize this?

Regular languages with rules

- The “regular expression” syntax is a shortcut representation
- We can describe the generative process more formally using a set of *rules* that are recursively applied

$$A \rightarrow aA$$

$$A \rightarrow a$$

terminal means “end”

- Rules have two types of symbols:
  - *nonterminal* symbols (capital letters) are recursively replaced until there are no more of them
  - *terminal* symbols (lowercase letters) are normal vocabulary items

With rules: all sequences of digits

- Alphabet: $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
- Language: $\mathscr{L}_1 = \Sigma^*$
- Generative rules: S → A

  A → 0A

  A → 1A

  A → 2A

  …

  A → 9A

  A → ϵ

S is a designated start symbol

Generation can pick any of these expansions

We finish by generating the empty string

Generate 0429

step	current string	apply rule
1	S	S → A
2	A	A → 0A
3	0A	A → 4A
3	04A	A → 2A
4	042A	A → 9A
5	0429A	A → ϵ
6	0429	DONE

With rules: whole numbers

- Alphabet: $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
- Language: $\mathscr{L}_2 = 0 \mid [1-9][0-9]*$
- Generative rules:

S → A	A → 0
S → B	C → 0C
B → 1C	C → 1C
…	…
B → 9C	C → 9C
	C → ϵ

Nonterminals capture "state" semantically. A="just a zero", B="leading nonzero digit", etc

"A 0, or a nonzero digit followed by zero or digits"

Generate 0

1	S	S → A
2	A	A → 0
3	0	DONE

Generate 429

1	S	S → B
2	B	B → 4C
3	4C	C → 2C
4	42C	C → 9C
5	429C	C → ϵ
6	429	DONE

Formal definition of a language

- Definitions: consider the set $(\Sigma, N, S \in N, R)$, where
  - Σ is the *vocabulary* which is a finite set of *terminal symbols*
  - N is a finite set of *nonterminals symbols*
  - $S \in N$ is a special nonterminal called the *start symbol*
  - $\alpha, \beta,$ *and* γ are *strings* of zero or more terminal and nonterminal symbols
  - R is a set of *rules* of the form $\alpha N \beta \rightarrow \gamma$

The Chomsky Hierarchy

- Named after Noam Chomsky, the MIT linguist
- Different constraints on the grammar rules changes the complexity of the kinds of languages that can be described
  - Language *generation* (rules → examples in the language) is always easy
  - Language *recognition* is harder (meaning, more compute-intensive) for higher languages
- More powerful languages are harder (meaning, more compute-intensive) to recognize

The Chomsky Hierarchy

Type	Rules	Name	Recognized by	Complexity
3	A → aB	regular	finite-state automata	$\mathcal{O}(n)$
2	A → α	context-free	pushdown automata	$\mathcal{O}(n^3)$
1	αAβ → αɣβ	context-sensitive	linear-bounded Turing machine	$\mathcal{O}(2^n)$
0	αAβ → ɣ	recursively enumerable	Turing machines	undecidable

- α, β, and ɣ are strings of zero or more terminals and nonterminals

Regular languages

- Definitions: consider the set $(\Sigma, N, S \in N, R)$, where
  - Σ is the *vocabulary* which is a finite set of *terminal symbols*
  - N is a finite set of *nonterminals symbols*
  - $S \in N$ is a special nonterminal called the *start symbol*
  - $\alpha, \beta,$ *and* γ are *strings* of zero or more terminal and nonterminal symbols
  - R is a set of *rules* of the form $\alpha N \beta \rightarrow \gamma$

Type	Rules	Name	Recognized by
3	A → aB	regular	finite-state automata

- All the languages we created earlier can be described with such rules

Context-free languages

- Definitions: consider the set $(\Sigma, N, S \in N, R)$, where
  - Σ is the *vocabulary* which is a finite set of *terminal symbols*
  - N is a finite set of *nonterminals symbols*
  - $S \in N$ is a special nonterminal called the *start symbol*
  - α, β, and γ are *strings* of zero or more terminal and nonterminal symbols
  - R is a set of *rules* of the form $\alpha N \beta \rightarrow \gamma$

Type	Rules	Name	Recognized by
2	$A \rightarrow \alpha$	Context-free	Pushdown automata

- This change might seem small, but it fundamentally alters the kinds of languages that can be generated

Context-free and not regular

- $\Sigma = \{a, b, c, \ldots, z\}$
- Create a context-free language for $\mathscr{L}$, the set of palindromes
- S→A

  A→aAa A → a

  A→bAb A → b

  … …

  A→zAz A → z

  A→ϵ
- Can you do this with the regular language constraint on rules?

Summary

- This view of languages (not sets, but “capturing” generative processes) is very productive
- We can generalize this discussion to make a connection between natural and other kinds of languages
- Consider, for example, *computer programs*
  - They either compile or don’t compile
  - *Their structure determines their interpretation*
- What is the structure?

Outline

formal language theory

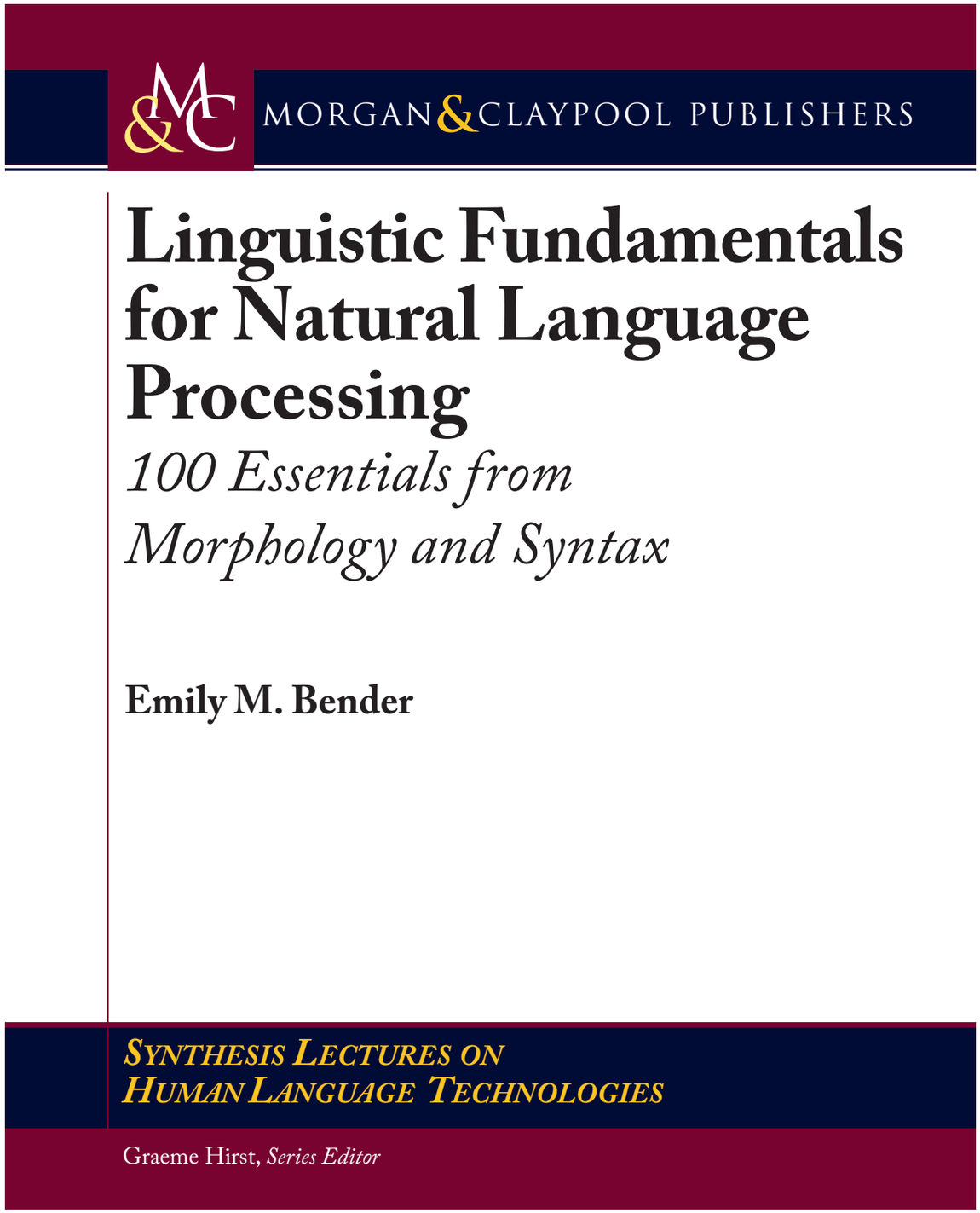
natural language

parsing

Linguistic fields of study

- Phonetics: sounds
- Phonology: sound systems
- Morphology: internal word structure
- **Syntax**: external word structure (sentences)
- Semantics: sentence meaning
- Pragmatics: contextualized meaning and communicative goals

Today's focus



- Excellent book
- Organized into 100 mini-lectures
- PDF available for free via JHU library (along with tens of others in the series)
- https://tinyurl.com/linguistic-fundamentals

What is syntax?

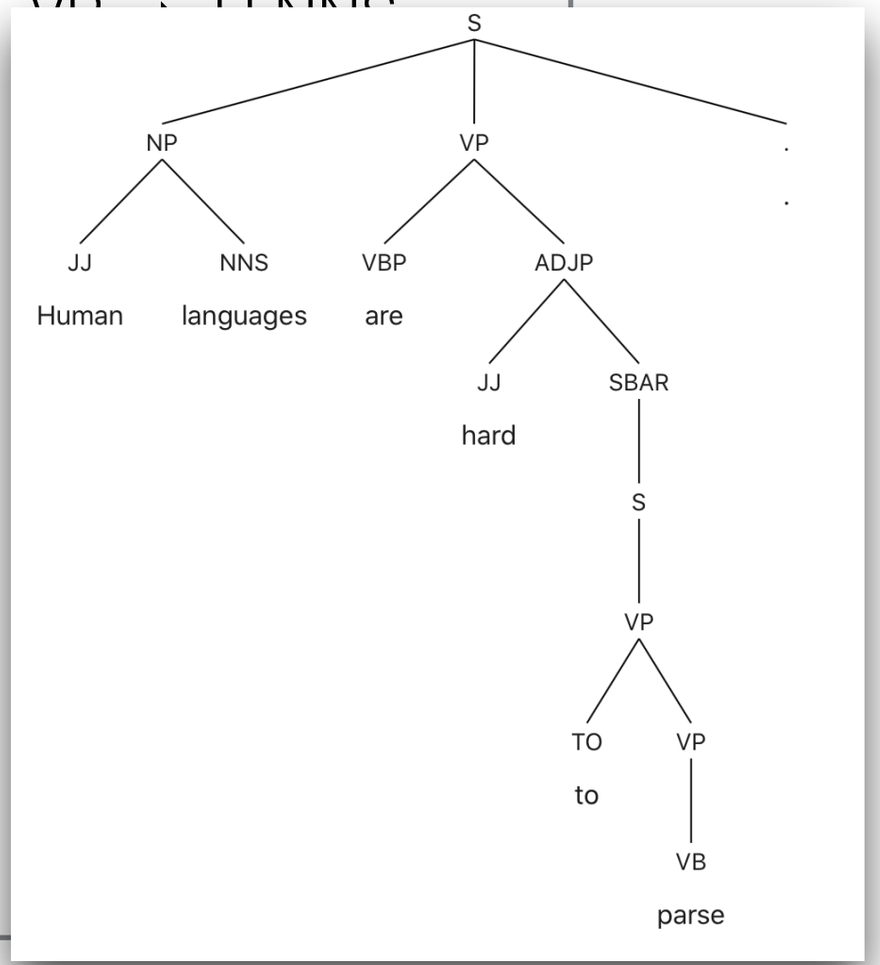
- A set of constraints on the possible sentences in the language
- * = “ungrammatical”
  - *A set of constraint on the possible sentence.
  - Dipanjan had *(a) question.
  - *You are on class.
- At a coarse level, we can divide all possible sequences of words into two groups: *valid* and *invalid* (or *grammatical* and *ungrammatical*)

Can you see the connection to formal language theory?

Example

S
S → NP VP .
S → [JJ NNS] VP .
S → [Human] NNS VP .
S → Human [languages] VP .
S → Human languages [VBP ADJP] .
S → Human languages [are] ADJP .
S → Human languages are [JJ SBAR] .
S → Human languages are [hard] SBAR .
S → Human languages are hard [VP] .
S → Human languages are hard [TO VP] .
S → Human languages are hard [to] VP .
S → Human languages are hard to [VB] .
S → Human languages are hard to [parse] .

S → NP VP



Nonterminals: parts of speech

- Functionally, parts of speech are unary rules that rewrite into words (the terminals)
- In grammar school, you may have learned "notional" definitions of parts of speech, e.g.,
  - noun: *a person, place, thing, or idea*
  - verb: *a word that shows action*
  - adjective: *a word that describes a noun*
- We typically instead use *distributional* definitions: parts of speech group words and phrases that can be substituted for each other

Phrases and Constituents

- Longer sequences of words can perform the same function as individual parts of speech:
  - I saw [a_{DT} kid_{N}]$_{NP}$
  - I saw [a kid playing basketball]$_{NP}$
  - I saw [a kid playing basketball alone on the court]$_{NP}$
- This gives rise to the idea of a **phrasal constituent**, which functions as a unit in relation to the rest of the sentence
- Constituents also can be grouped semantically with a nonterminal:
  - noun phrases, verb phrases, prepositional phrases, etc

Constituent tests

- How do you know if a phrase functions as a constituent?
- A few tests
  - *Coordination*
    - Kim [read a book], [gave it to Sandy], and [left].
  - *Substitution with a word*
    - Kim read [a very interesting book about grammar].
    - Kim read [it].
  - See Bender #51

What *isn't* syntax?

- Syntax is a scaffolding for meaning, but is not the same as meaning

grammatical & meaningful	grammatical & meaningless
ungrammatical & meaningful	ungrammatical & meaningless

I am here

Slick flat anemones drily type

You is talls

very pancake up

Human judgments

- "Ungrammatical" = "not in the language"
- Humans play a unique role in this
  - How do we know what's grammatical? We ask native speakers to make judgments
- But how do native speakers know? We don't know…
  - …but we can use formal language theory to try to explain it
  - syntactic theories are grammars

Language as a CFG

- e.g., some rules
  - [sentence] → [subject] [predicate]
  - [subject] → [noun phrase]
  - [noun phrase] → [determiner]? [adjective]* [noun]
  - [predicate] → [verb phrase] [adjunct]
- Rules are *phrasal* or *terminal*
  - Phrasal rules form **constituents** in a tree
  - Terminal rules are **parts of speech** and produce words

Chomsky formal language hierarchy refresher

regular grammar
context-free grammar
context-sensitive grammar
Turing machine

Treebanks

- Collections of natural text that are annotated according to a particular syntactic theory
  - Usually created by linguistic experts
  - Ideally as large as possible
  - Theories are usually coarsely divided into *constituent/phrase* or *dependency* structure (not discussed)

The Penn Treebank

- Syntactic annotation of a million words of the 1989 Wall Street Journal, plus other corpora (released in 1993)
  - (Trivia: People often discuss “The Penn Treebank” when the mean the WSJ portion of it)
- Contains 74 total tags: 36 parts of speech, 12 punctuation and currency tags, and 31 phrasal constituent tags, plus some relation markings
- Was the foundation for an entire field of research and applications for over twenty years

Generative story

- The “generative story” is that a grammar was used to generate every sentence
- Linguists annotate the parse trees, and we can read this “story” as a CFG that produced each sentence
- We can extract all rules over the entire Treebank and use them to determine the “language” of English
  - This is the connection between formal language theory and linguistic syntax

```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD                    S years) )
        (JJ old)
      (, ,) )
    (VP (MD wil
      (VP (VB join)
    (NP (DT the) (NN board) )
    (PP-CLR (IN as)
      (NP (DT a) (JJ nonexecutive) (NN director) ))
    (NP-TMP (NNP Nov.) (CD 29) )))
(. .) ))
```

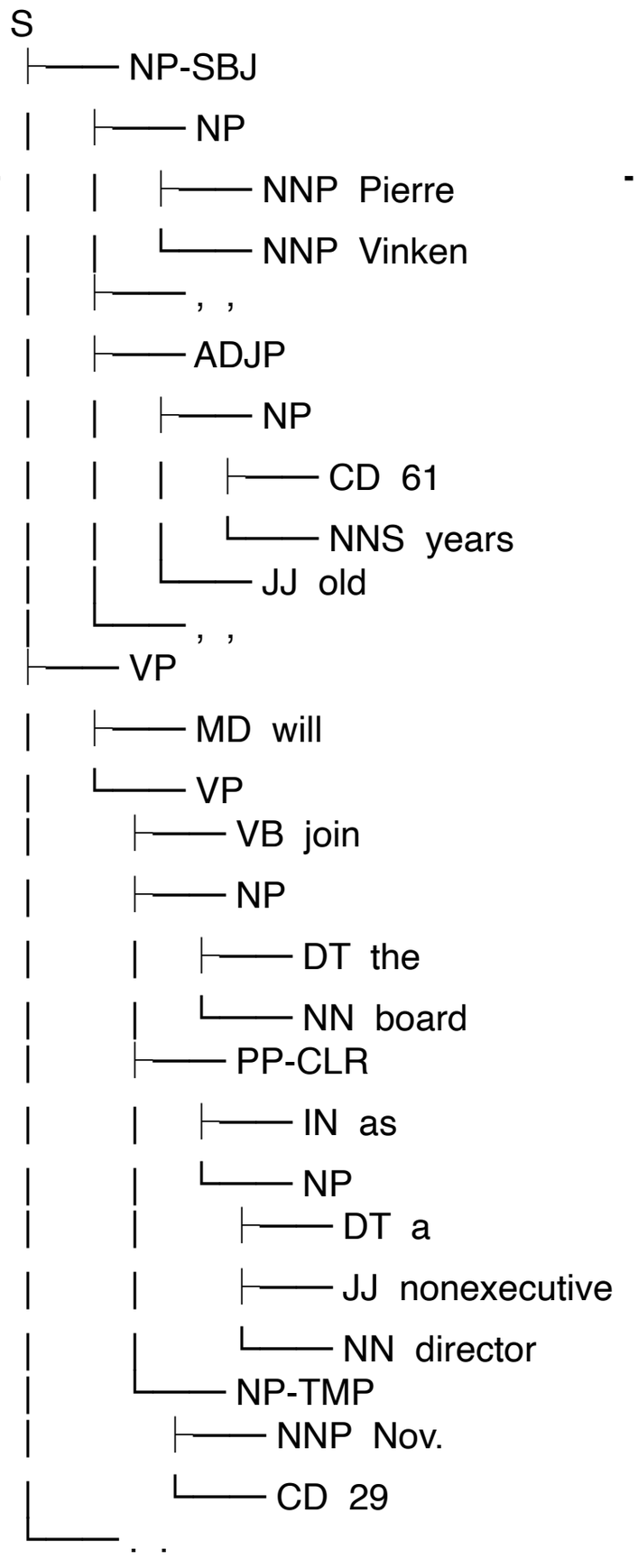
x 49,208

https://commons.wikimedia.org/wiki/File:PierreVinken.jpg

Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.

Rule extraction

- Here the parse has been rewritten for readability
- Rules can be read from the indentation, e.g.,
  S → NP-SBJ VP .
  NP-SBJ → NP , ADJP ,
  NP → NNP NNP
  NNP → Pierre
  NNP → Vinken
  VP → MD VP
  VP → VB NP PP-CLR NP-TMP
  …



Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.

Some notes

- Nonterminals are finer-grained, e.g., NP-TMP
- This isn't perfect
  - There are thousands and thousands of rules
  - The grammar over-generates wildly, including many invalid English sentences
- Not discussed: probabilities
  - We can annotate all rules with a probability distribution
  - For example, the sum of all rules S → ? will sum to 1
  - Relative frequencies can be read from the data

Where does English fall?

- How much expressive power do we need to model English?
- There are languages the exhibit non context-free phenomena (Swiss-German and it's cross-serial dependencies, notoriously), but we typically use CFGs for efficiency reasons

Chomsky formal language hierarchy refresher

regular grammar
context-free grammar (cubic-time parsing)
context-sensitive grammar
Turing machine

Summary so far

- Formal language theory is a theory that does the following:
  - provides a compact representation of a language
  - provides an account for how strings within a language are generated
- It's very useful for describing many simple languages
- It can also be applied to natural language
- Treebanks were used to annotate real data according to theories of syntax

Outline

formal language theory

natural language

parsing

Where we are

- We discussed formal language theory
- We showed how it might apply to human language
- But how do we get a computer to use it?
  - Sentences (or other strings we wish to parse) are observed; the structure is *latent* (hidden) and needs to be inferred
- Assume: the text was generated by a model
  - We need
    - An algorithm for finding the sequence of actions under that model, most likely to have produced it
    - A way to learn that model

Parsing

- If the grammar has certain properties, we can efficiently answer the first question (find the hidden structure) with a **parser**
  - Q1: is the sentence in the language of the grammar?
  - Q2: *what* is the structure above that sentence?

Important properties

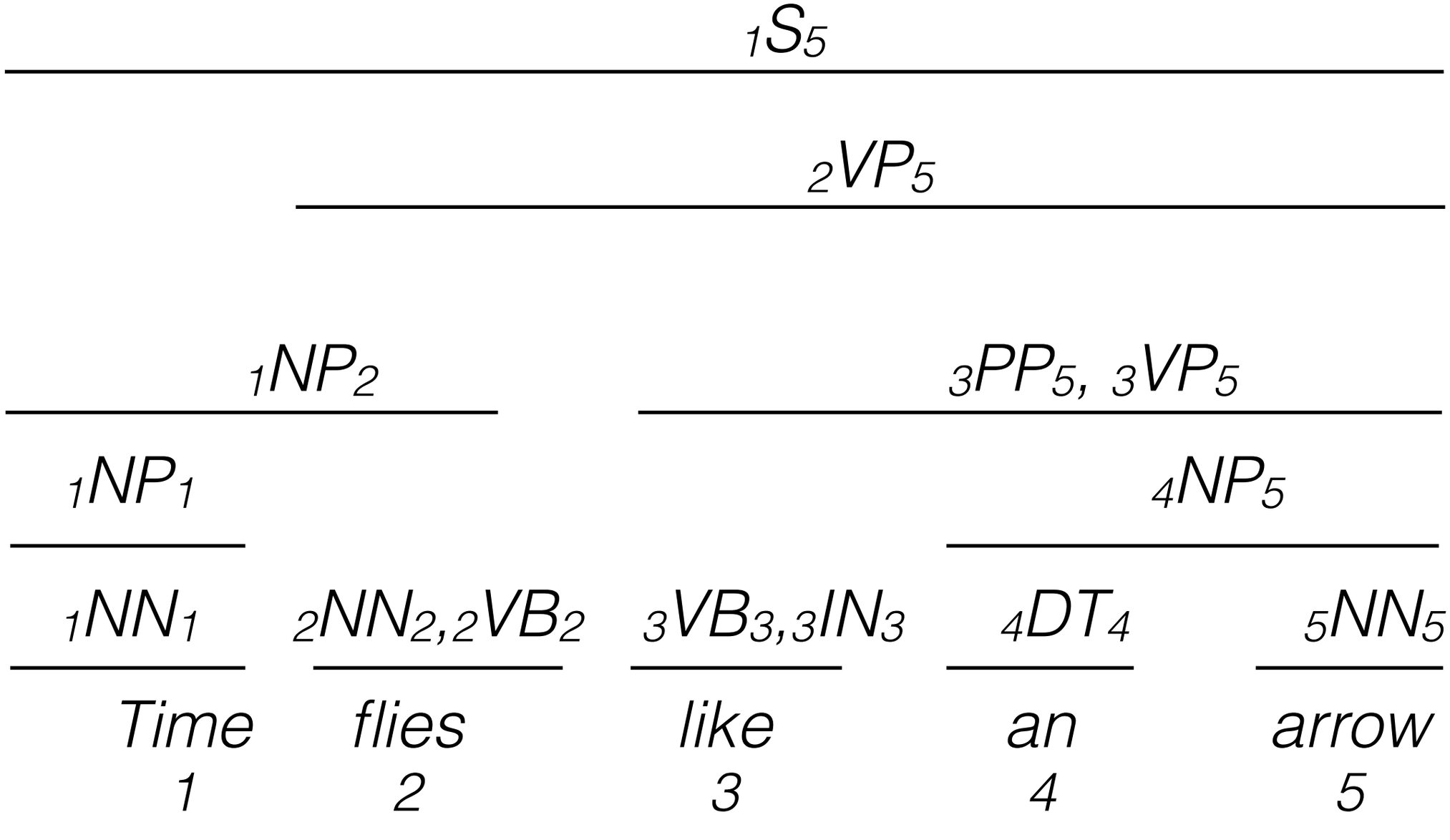
- *Chomsky Normal Form*
  - Every rule is either binary (S → A B)
    or terminal (A → a)
  - Longer rules must be binarized, e.g., S → A B C
    S → A X_1
    X_1 → B C
- We need special handling for unary rules to prevent infinite inference loops
  - If the grammar has A → B *and* B → A, we need to take care not to infer a (useless) infinite chain
    A → B → A → B → A → …

Chart parsing for constituency grammars

- Maintains a chart of nonterminals spanning words, e.g.,
  - NP over words 1..4 and 2..5
  - VP over words 4..6 and 4..8
  - etc
- If we can build an S over the whole sentence, the sentence is in the language

Chart parsing for constituency grammars

If we can build S over the whole sentence, the sentence is **in** the grammar



Cocke-Kasami-Younger algorithm

- Basic idea: work bottom-up, applying all matching rules, (recursively) building larger constituents from smaller one

```
# base case
for i in 1..N
  for all rules A → wᵢ
    create ᵢAᵢ

# inductive case
for width in 2..N
  for i in 1..{N - width + 1}
    j = i + width - 1
    for split k in i..{j - 1}
      for all rules A → B C
        create ᵢAⱼ if ᵢBₖ and ₍ₖ₊₁₎Cⱼ
```

Note the normal form (binary rule) assumption

Not handled here: unary rules

CKY algorithm

The two decompositions correspond to different semantics

$S \rightarrow {}_1NP_2\ {}_3VP_5$ — Funny meaning

$S \rightarrow {}_1NP_1\ {}_2VP_5$ — Normal meaning

$VP \rightarrow VB\ PP$

$VP \rightarrow {}_3VB_3\ {}_4NP_5$

$PP \rightarrow {}_3IN_3\ {}_4NP_5$

$NP \rightarrow NN\ NN$

$NP \rightarrow NN$

$NP \rightarrow DT\ NN$

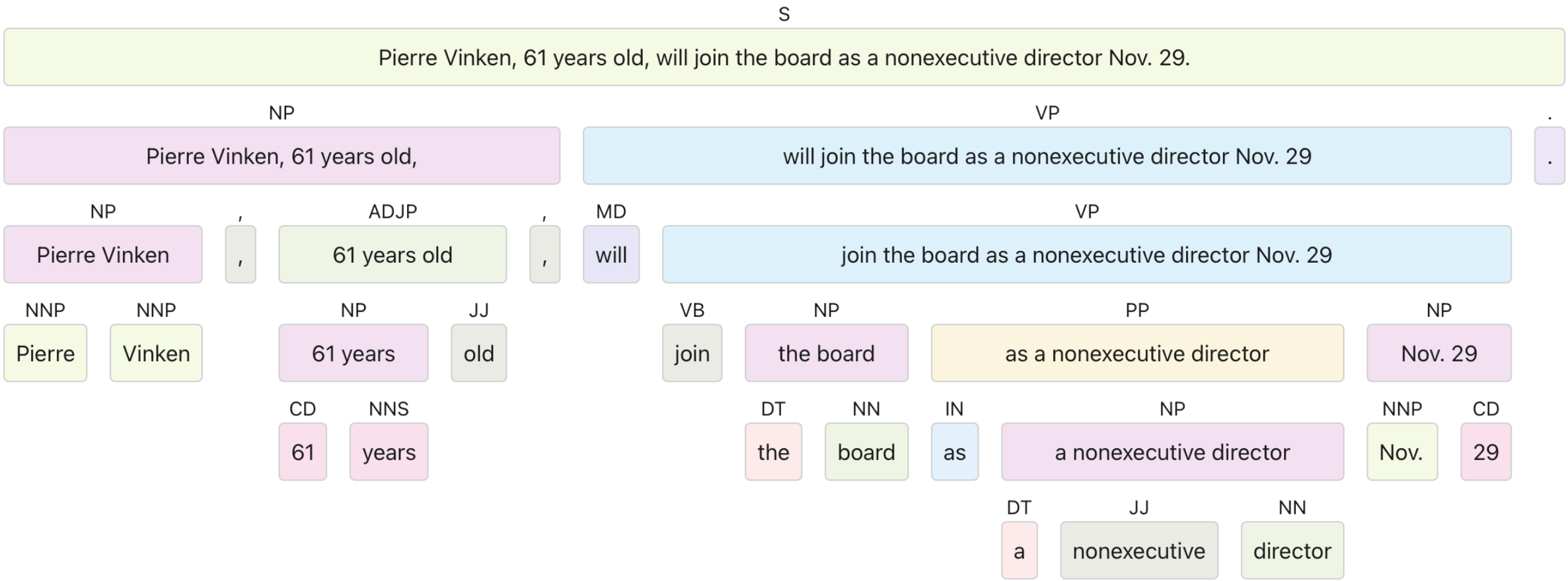
NN	NN,VB	VB,IN	DT	NN
Time	flies	like	an	arrow
1	2	3	4	5

CKY algorithm

- Parsing questions:
  - **Q1**: is a given sentence in the language of the grammar?
  - **Q2**: What is the structure above that sentence?
- Termination: is there a chart entry at $_1S_N$?
  - ✓ string is in the language (Q1)
  - Structures can be obtained by following backpointers in dynamic programming chart (Q2, not covered today)
- Other technical details not covered today:
  - We can disambiguate by assigning probabilities to rules

Demos

- Berkeley Neural Parser:
  https://parser.kitaev.io/
  (certificate expired, but seems okay…)



Summary

formal language theory	natural language	parsing
provides a framework for reasoning about languages of all kinds	*a real-world (if messy) application area for formal language theory*	*a means of making text useable under formal language theory*