CS 490: NATURAL LANGUAGE PROCESSING

Dan Goldwasser, Abulhair Saparov

Lecture 11: Morphology and Syntax

SUBWORD TOKENIZATION

- How do we tokenize at the right level of granularity?
  - One approach is to train a tokenizer from data.
  - This is the approach taken by byte pair encoding (BPE; Sennrich et al., 2016).

- In BPE, we start with a vocabulary containing all individual characters.

$$\Sigma = \{\text{‘A’},\text{‘B’},\text{‘C’},\ldots,\text{‘Y’},\text{‘Z’},\text{‘a’},\text{‘b’},\text{‘c’},\ldots,\text{‘y’},\text{‘z’},\text{‘0’},\text{‘1’},\ldots\}$$

- Then repeat k times:
  - Choose the two symbols in Σ that occur most frequently together in the training corpus (e.g., ‘u’ and ‘n’).
  - Add a new merged symbol (e.g, ‘un’) to Σ.
  - Replace all adjacent ‘u’ and ‘n’ in the training corpus with ‘un’.

BYTE PAIR ENCODING

- Some preprocessing is typically done with BPE:
    - The corpus is split into words by spaces.
    - The space is added to the end of each word as a special token,
        - E.g., ‘`the stars shone`’ -> [‘`the_`’, ‘`stars_`’, ‘`shone`’].
- Once we have a learned vocabulary, we can tokenize any new text:
    - Perform each merge operation in the same order that it was learned during training.
- BPE is used in all major LLMs.

WORDPIECE TOKENIZATION

- An alternative subword tokenization method is WordPiece tokenization.
- It is largely identical to BPE, with the core difference being the merge rule:
- In BPE, at each iteration, the two symbols that most frequently appear together in the training corpus are merged.
- In WordPiece, we instead select the two symbols a and b that maximize the quantity:

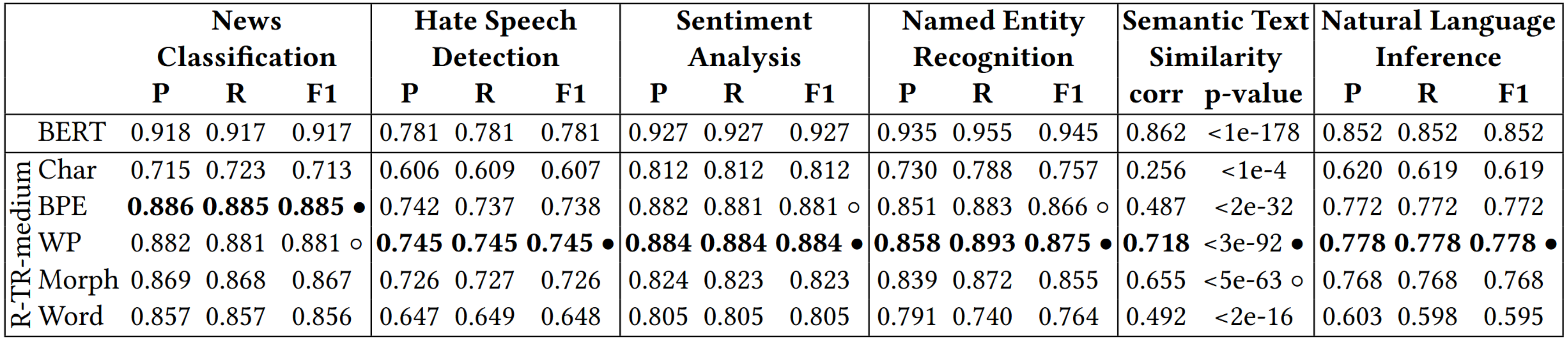
$$\frac{\texttt{frequency}(ab)}{\texttt{frequency}(a)\cdot\texttt{frequency}(b)}.$$

WORDPIECE TOKENIZATION

- Once trained, the WordPiece tokenizer also differs slightly from the trained BPE tokenizer.
- Instead of applying merge operations in the same order in which they were learned,
- The WordPiece tokenizer simply matches tokens greedily.
  - Starting from the beginning of the sequence, it finds the longest subword in its vocabulary that matches the input.
  - Then it repeats.

EFFECT OF TOKENIZATION

- Toraman et al. (2022) empirically tested the effect of different tokenizers on the performance of medium-sized RoBERTa models.
  - They focused on Turkish, which is very morphologically rich (i.e., words often contain many affixes).



		News Classification			Hate Speech Detection			Sentiment Analysis			Named Entity Recognition			Semantic Text Similarity		Natural Language Inference		
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**corr**	**p-value**	**P**	**R**	**F1**
	BERT	0.918	0.917	0.917	0.781	0.781	0.781	0.927	0.927	0.927	0.935	0.955	0.945	0.862	<1e-178	0.852	0.852	0.852
R-TR-medium	Char	0.715	0.723	0.713	0.606	0.609	0.607	0.812	0.812	0.812	0.730	0.788	0.757	0.256	<1e-4	0.620	0.619	0.619
	BPE	**0.886**	**0.885**	**0.885** ●	0.742	0.737	0.738	0.882	0.881	0.881 ○	0.851	0.883	0.866 ○	0.487	<2e-32	0.772	0.772	0.772
	WP	0.882	0.881	0.881 ○	**0.745**	**0.745**	**0.745** ●	**0.884**	**0.884**	**0.884** ●	**0.858**	**0.893**	**0.875** ●	**0.718**	<3e-92 ●	**0.778**	**0.778**	**0.778** ●
	Morph	0.869	0.868	0.867	0.726	0.727	0.726	0.824	0.823	0.823	0.839	0.872	0.855	0.655	<5e-63 ○	0.768	0.768	0.768
	Word	0.857	0.857	0.856	0.647	0.649	0.648	0.805	0.805	0.805	0.791	0.740	0.764	0.492	<2e-16	0.603	0.598	0.595

WHAT ABOUT OTHER LANGUAGES?

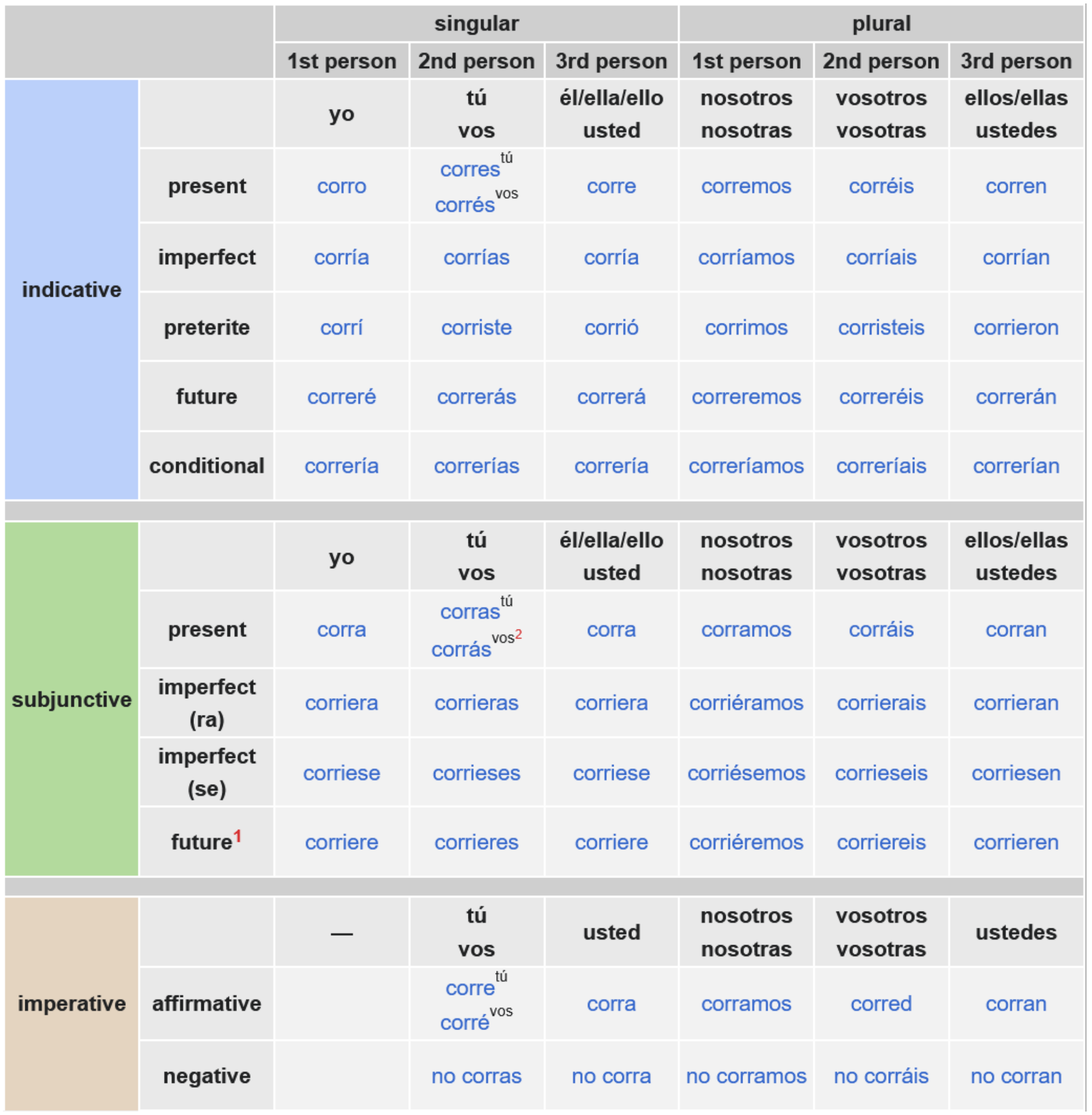
- Not all languages have morphologies similar to English.
- Some languages have little to no morphology.
  - These are called isolating or analytic languages.
- E.g., Yoruba, Vietnamese
- E.g., Chinese
  - There are some examples of inflection (e.g., ‘们’ or ‘mén’ can denote plural),
  - As well as some examples of derivation (e.g., ‘艺术家’ or ‘yìshùjiā’ which means ‘artist’).
  - But these are rare compared to other languages.
- Chinese contains a significant number of compound words (~80% of Chinese words are compounds).

WHAT ABOUT OTHER LANGUAGES?

- In fusional languages, each affix can encode information about multiple grammatical features.
- English has some examples of "fusion" but not as much as other languages (Modern English has become more analytic relative to earlier forms).
  - E.g., '`-es`' in '`crosses`' denotes 3rd person, singular, and present tense.
- E.g., Most Indo-European languages: French, Spanish, Italian, Greek, Irish, Polish, Russian, Ukranian, Lithuanian, etc.

  (proto-Indo-European was most likely a fusional language)

FUSIONAL MORPHOLOGY IN SPANISH



		singular			plural		
		1st person	2nd person	3rd person	1st person	2nd person	3rd person
indicative		yo	tú vos	él/ella/ello usted	nosotros nosotras	vosotros vosotras	ellos/ellas ustedes
	present	corro	corres$^{\text{tú}}$ corrés$^{\text{vos}}$	corre	corremos	corréis	corren
	imperfect	corría	corrías	corría	corríamos	corríais	corrían
	preterite	corrí	corriste	corrió	corrimos	corristeis	corrieron
	future	correré	correrás	correrá	correremos	correréis	correrán
	conditional	correría	correrías	correría	correríamos	correríais	correrían
subjunctive		yo	tú vos	él/ella/ello usted	nosotros nosotras	vosotros vosotras	ellos/ellas ustedes
	present	corra	corras$^{\text{tú}}$ corrás$^{\text{vos}}$[2]	corra	corramos	corráis	corran
	imperfect (ra)	corriera	corrieras	corriera	corriéramos	corrierais	corrieran
	imperfect (se)	corriese	corrieses	corriese	corriésemos	corrieseis	corriesen
	future[1]	corriere	corrieres	corriere	corriéremos	corriereis	corrieren
imperative		—	tú vos	usted	nosotros nosotras	vosotros vosotras	ustedes
	affirmative		corre$^{\text{tú}}$ corré$^{\text{vos}}$	corra	corramos	corred	corran
	negative		no corras	no corra	no corramos	no corráis	no corran

[https://en.wiktionary.org/wiki/correr#Spanish]

TEMPLATE-BASED FUSIONAL MORPHOLOGY

- Affixes are not always added to the beginning or ends of roots.
- In Semitic languages (i.e., Akkadian, Arabic, Aramaic, Hebrew, Phoenician), the roots of words are three consonants.
  - Triconsonantal roots
- Words can be constructed by adding different vowels between the consonants.
  - **k**a**t**a**b**a كَتَبَ or كتب "he wrote"
  - **k**a**t**a**b**at كَتَبَت or كتبت "she wrote"
  - **k**i**t**ā**b** كِتَاب or كتاب "book"
  - ma**k**ta**b** مَكتَب or مكتب "desk" or "office"
  - ma**kt**a**b**at مَكتَبَة or مكتبة "library" or "bookshop"

AGGLUTINATIVE MORPHOLOGY

- In some languages, each affix encodes information about a single grammatical feature.
  - Multiple affixes can be chained together in a linear and systematic fashion.
- This is called agglutination.
- Examples of agglutinative languages: Finnish, Japanese, Korean, Swahili.

AGGLUTINATIVE MORPHOLOGY

- An extreme example of agglutination from Turkish:

uygarlaştıramadıklarımızdanmışsınızcasına

"(behaving) as if you are among those whom we were not able to civilize"

uygar	"civilized"
+laş	"become"
+tır	"cause to"
+ama	"not able"
+dık	past participle
+lar	plural
+ımız	first person plural possessive ("our")
+dan	ablative case ("from/among")
+mış	past
+sınız	second person plural ("y' all")
+casına	finite verb → adverb ("as if")

[Olfazer and Güzey, 1994]

POLYSYNTHETIC MORPHOLOGY

- Some languages can utilize morphology to encode the meaning of full sentences.
- E.g., many (but not all) Native American languages, Ainu, Mayan, Quechua.
    - E.g., in Yupik: ‘tuntussuqatarniksaitengqiggtuq’

tuntu	-ssur	-qatar	-ni	-ksaite	-ngqiggte	-uq
“reindeer”	“hunt”	[future]	“say”	[negative]	“again”	[3rd person, singular, indicative]

-
    - Means: “He had not yet said again that he was going to hunt reindeer.”
    - Only ‘tuntu’ can stand alone as a word.
- Verbs can be attached to nouns as affixes.

COMPUTATIONAL LINGUISTICS ROADMAP

- In this lecture, we discussed computational linguistics, at a high level.
- No need to memorize different morphologies of different languages.
  - Key takeaway:
  - Be aware of the diversity of morphologies of languages around the world.
  - And how they may affect NLP methods.
  - Certain NLP methods may work better for some languages than others.
- Next time: Syntax
  - How are words arranged to form sentences?
  - What is a grammar?
  - Syntactic composition
- Later: Semantics

SYNTAX

SYNTAX

- Consider the sentence ‘`Sam ran to the bank.`’
- The order of the words has a significant effect on the grammaticality and meaning of the sentence.
- Some orderings produce ungrammatical sentences: ‘`Ran to Sam the bank.`’
- Other orderings change the meaning/semantics: ‘`The bank ran to Sam.`’
- Other orderings produce grammatical sentences that preserve the meaning:
  - ‘`To the bank, ran Sam.`’
  - ‘`To the bank, Sam ran.`’
  - ‘`Sam, to the bank, ran.`’
  - Notice these sentences maintain three contiguous phrases: ‘`Sam`’, ‘`ran`’, ‘`to the bank`’.

SYNTAX

- Another way to understand syntax is to consider substitutions of words and/or phrases.
  - What can we substitute for ‘`Sam`’ while preserving grammaticality?
  - ‘`A cat ran to the bank`’ (grammatical)
  - ‘`Fast ran to the bank`’ (ungrammatical, unless ‘`Fast`’ is a name)
  - The words ‘`Sam`’ and ‘`a cat`’ belong to the same grammatical category.
    - They are nouns.
  - But ‘`fast`’ is in a different grammatical category.
    - An adjective or adverb.

PARTS OF SPEECH

- These grammatical categories are called parts of speech (POS).
  - Nouns: ‘cat’, ‘bank’, ‘Sam’
  - Verb: ‘run’, ‘sleep’, ‘is’
  - Adjectives: ‘fast’, ‘orange’, ‘short’
  - Adverbs: ‘quickly’, ‘probably’, ‘slowly’
  - Prepositions: ‘to’, ‘in’, ‘of’
  - Determiners: ‘a’, ‘the’, ‘those’
  - Pronouns: ‘I’, ‘me’, ‘mine’
- Some words can have multiple parts of speech:
  - ‘The fast car drove by.’
  - ‘The car drove by fast.’

PARTS OF SPEECH

- Parts of speech can vary across languages.
  - Many East Asian Languages have measure words or classifiers.
  - E.g., ‘버스 티켓 열 장’

버스	티켓	열	장
"bus"	"ticket"	"10"	[measure word for sheet-like objects]

  - ‘피자 한 조각’

피자	한	조각
"pizza"	"1"	[measure word for slices]

- Some languages use postpositions:
  - Similar to prepositions, except the postposition occurs after the noun phrase.
  - E.g., in Hungarian: ‘`fa alatt`’ is literally ‘tree under’ but means ‘under the tree’.

PARTS OF SPEECH

- Some parts of speech are closed classes; new words are rarely added.
  - Determiners, prepositions, postpositions, pronouns, measure words
- Others are open classes; and new words are added readily.
  - Nouns, verbs, adjectives, adverbs
- Closed classes do change, but typically over much longer time periods.
  - E.g., the preposition ‘`during`’ comes from the verb ‘`dure`’ (‘`dure`’+‘`ing`’).
    - Probably before or around the time of Middle English.
  - The verb ‘`dure`’ means to continue, to endure, to last.

SYNTAX

- We can also substitute words with phrases:
  - ‘`The fast cat with the stripes ran to the bank`’ (grammatical)
  - ‘`The fast cat with the stripes`’ is a phrase that behaves like a noun.
    - This is a noun phrase.
- We can similarly define other kinds of phrases:
  - Verb phrase: ‘`ran to the bank`’
  - Adjective phrase: ‘`taller than a mountain`’
  - Adverbial phrase: ‘`very quickly`’
  - Prepositional phrase: ‘`to the bank`’

SYNTAX

- Focusing on the phrase '`to the bank`', each component is necessary to maintain the grammaticality and meaning of the sentence.
  - '`Sam ran bank`' (ungrammatical)
  - '`Sam ran to bank`' (ungrammatical)
  - '`Sam ran the bank`' (grammatical, but different meaning)
  - All three words are needed to form the phrase '`to the bank.`'
- But looking closely at the prepositional phrase, we see that it is composed of smaller components:
  - A preposition ('`to`') and a noun phrase ('`the bank`').
  - We can conclude that one way to construct prepositional phrases is to combine any preposition with a noun phrase.

SYNTAX

- We can also substitute words with phrases:
  - ‘`The fast cat with the stripes ran to the bank`’ (grammatical)
  - ‘`The fast cat with the stripes`’ is a phrase that behaves like a noun.
    - This is a noun phrase.
- We can similarly define other kinds of phrases:
  - Verb phrase: ‘`ran to the bank`’
  - Adjective phrase: ‘`taller than a mountain`’
  - Adverbial phrase: ‘`very quickly`’
  - Prepositional phrase: ‘`to the bank`’

SYNTAX

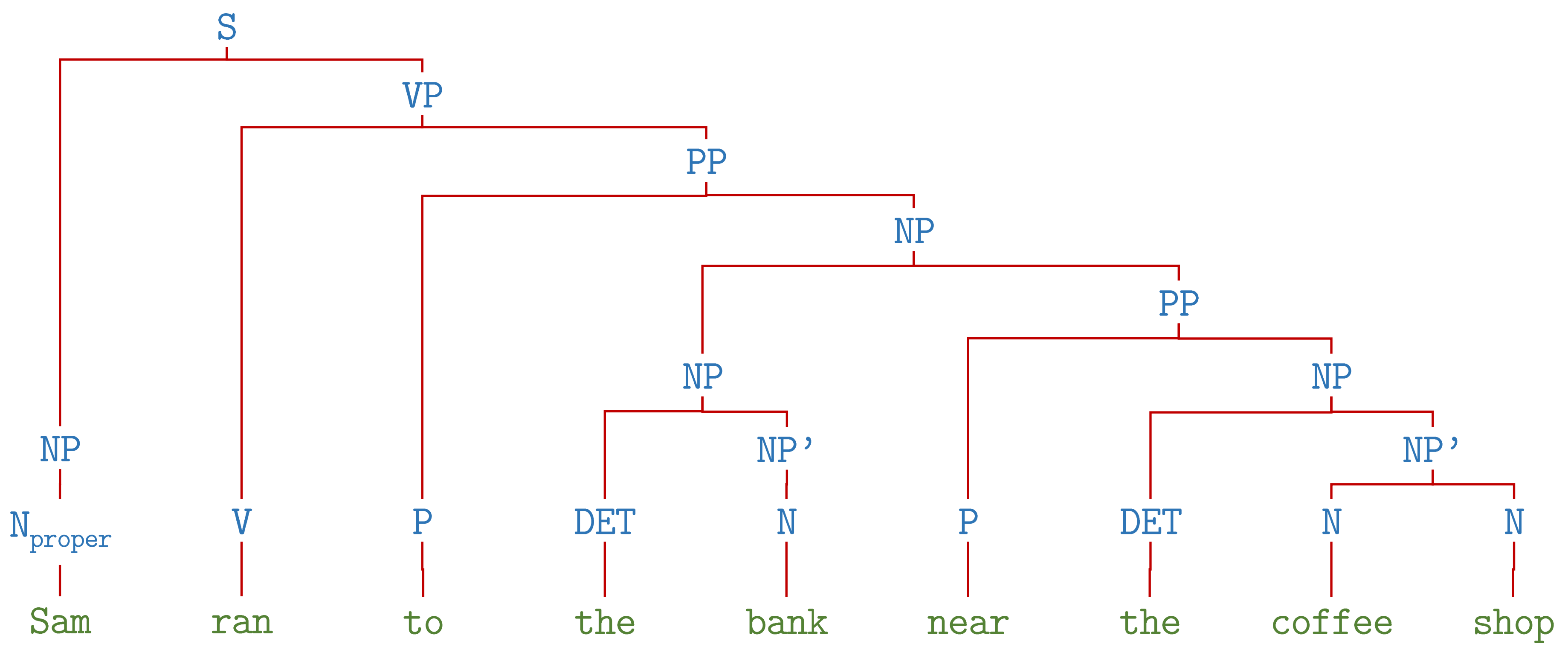
- Focusing on the phrase ‘`to the bank`’, each component is necessary to maintain the grammaticality and meaning of the sentence.
    - ‘`Sam ran bank`’ (ungrammatical)
    - ‘`Sam ran to bank`’ (ungrammatical)
    - ‘`Sam ran the bank`’ (grammatical, but different meaning)
    - All three words are needed to form the phrase ‘`to the bank.`’
- But looking closely at the prepositional phrase, we see that it is composed of smaller components:
    - A preposition (‘`to`’) and a noun phrase (‘`the bank`’).
    - We can conclude that one way to construct prepositional phrases is to combine any preposition with a noun phrase.

SYNTAX

- For example, we can rephrase the sentence as '`Sam ran to Chase Bank.`'
- So '`Chase Bank`' and '`the bank`' play the same grammatical role.
  - They are noun phrases.
- Language enables to construction of larger noun phrases:
  - E.g., '`Sam ran to the bank near the coffee shop.`'
  - Here, we made a larger noun phrase by adding the prepositional phrase "near the coffee shop."
- So we have seen that prepositional phrases can contain subordinate noun phrases, and noun phrases can contain subordinate prepositional phrases.
  - This is an example of recursion,
  - Recursion is a key property of languages, including natural language.

SYNTAX TREE

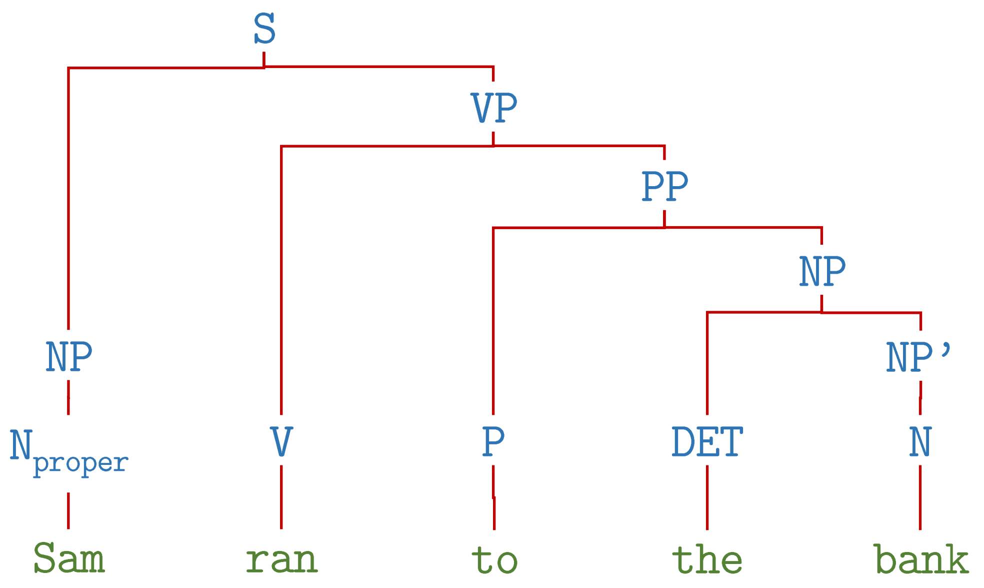
- A natural way to represent recursive structure is as a tree:



SYNTAX TREE

- Syntax trees can also be represented in bracketed notation:

```
(S
    (NP (N_proper ‘Sam’))
    (VP
        (V ‘ran’)
        (PP
            (P ‘to’)
            (NP
                (DET ‘the’)
                (NP’ (N ‘bank’))
            )
        )
    )
)
```



GRAMMAR

- We have observed that noun phrases can be constructed from smaller phrases:
  - Either a proper noun (e.g., ‘`Sam`’),
  - A determined common noun (e.g., ‘`the bank`’),
  - Or a noun phrase + prepositional phrase (e.g. ‘`the bank near me`’),
  - (or other rules)
- Similarly, we have observed prepositional phrases can be constructed from smaller phrases.
  - And similarly for verb phrases, etc.
- These are grammatical rules.
  - Altogether, these rules form the grammar of a language.

GRAMMAR

- We can write the rules of a grammar:
- Each rule is called a production rule.
- S is the root.

S -> NP VP	V -> ‘ran’	N_{proper} -> ‘Ada’
VP -> V NP	V -> ‘sees’	N_{proper} -> ‘Alex’
VP -> V PP	V -> ‘see’	N_{proper} -> ‘Sam’
PP -> P NP	P -> ‘to’	N_{proper} -> ‘Zach’
NP -> NP PP	P -> ‘near’	DET -> ‘the’
NP -> N_{proper}	N -> ‘bank’	DET -> ‘a’
NP -> DET NP’	N -> ‘coffee’	
NP’ -> N	N -> ‘shop’	

GRAMMAR

- Nonterminals: S, NP, VP, PP, NP’, DET, N_{proper}, N, V, P
- Terminals: ‘ran’, ‘sees’, ‘see’, ‘to’, …, ‘Ada’, ‘Alex’, …
- Preterminals: DET, N_{proper}, N, V, P (this is a subset of nonterminals)

```
S   -> NP VP          V -> ‘ran’       N_proper -> ‘Ada’
VP ->  V NP           V -> ‘sees’      N_proper -> ‘Alex’
VP ->  V PP           V -> ‘see’       N_proper -> ‘Sam’
PP ->  P NP           P -> ‘to’        N_proper -> ‘Zach’
NP -> NP PP           P -> ‘near’      DET -> ‘the’
NP ->  N_proper       N -> ‘bank’      DET -> ‘a’
NP -> DET NP’         N -> ‘coffee’
NP’ -> N              N -> ‘shop’
```

GRAMMAR

- This grammar specifies a set of strings (i.e., a language).
- To sample a string from the grammar, start with the root S.
  - Recursively: For any nonterminal in the input, pick a rule with a matching left-hand side. Rewrite the nonterminal with the right-hand side.

S -> NP VP	V -> ‘ran’	N_{proper} -> ‘Ada’
VP -> V NP	V -> ‘sees’	N_{proper} -> ‘Alex’
VP -> V PP	V -> ‘see’	N_{proper} -> ‘Sam’
PP -> P NP	P -> ‘to’	N_{proper} -> ‘Zach’
NP -> NP PP	P -> ‘near’	DET -> ‘the’
NP -> N_{proper}	N -> ‘bank’	DET -> ‘a’
NP -> DET NP’	N -> ‘coffee’	
NP’ -> N	N -> ‘shop’	

GRAMMAR

- This grammar specifies a set of strings (i.e., a language).
- To sample a string from the grammar, start with the root S.
  - Recursively: For any nonterminal in the input, pick a rule with a matching left-hand side. Rewrite the nonterminal with the right-hand side.
- Example:
  1. S
  2. NP VP
  3. $\mathtt{N}_{\mathtt{proper}}$ VP
  4. Sam VP
  5. Sam V NP
  6. Sam sees NP
  7. Sam sees $\mathtt{N}_{\mathtt{proper}}$
  8. Sam sees Ada

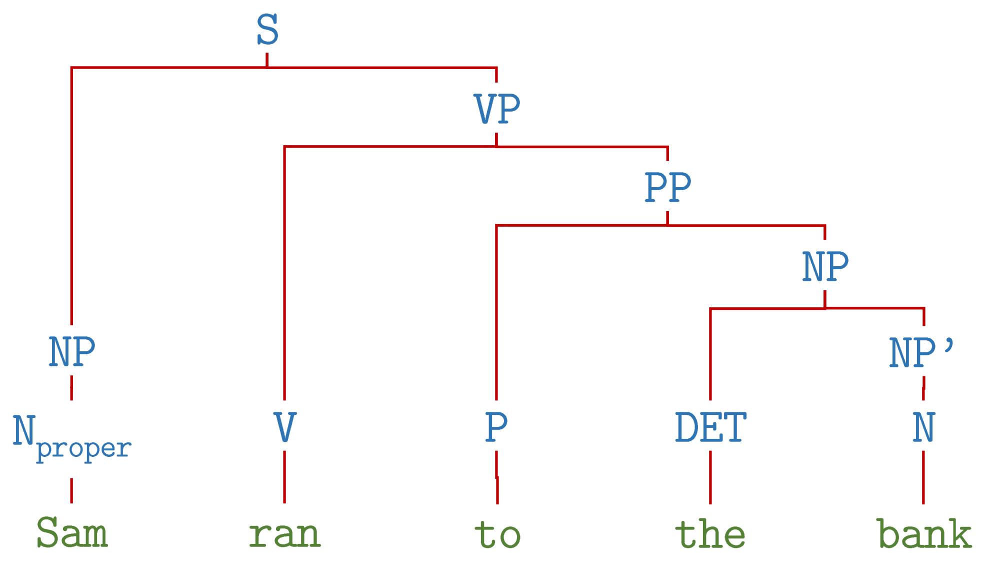
GRAMMAR

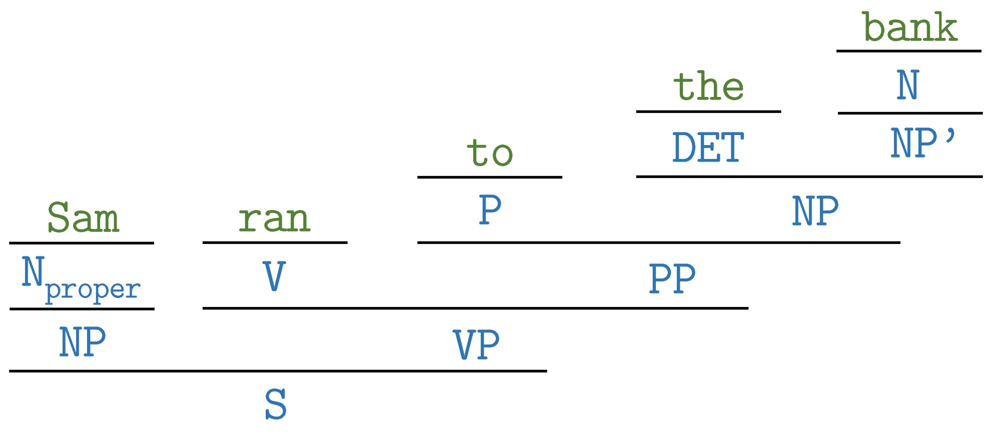
- This grammar specifies a set of strings (i.e., a language).
- To sample a string from the grammar, start with the root S.
    - Recursively: For any nonterminal in the input, pick a rule with a matching left-hand side. Rewrite the nonterminal with the right-hand side.
- The language generated by a grammar is the set of all strings s that can be generated with the above procedure.
- In this example grammar, the generated language is infinite:

```
L = {‘Sam sees Ada’, ‘Sam see Ada’, ‘Alex sees the coffee’, ‘Zach ran the shop’,
     ‘Sam ran to the bank’, ‘Sam ran to the bank near the shop’,
     ‘Sam ran to the bank near the shop near Ada’, …}
```

GRAMMAR

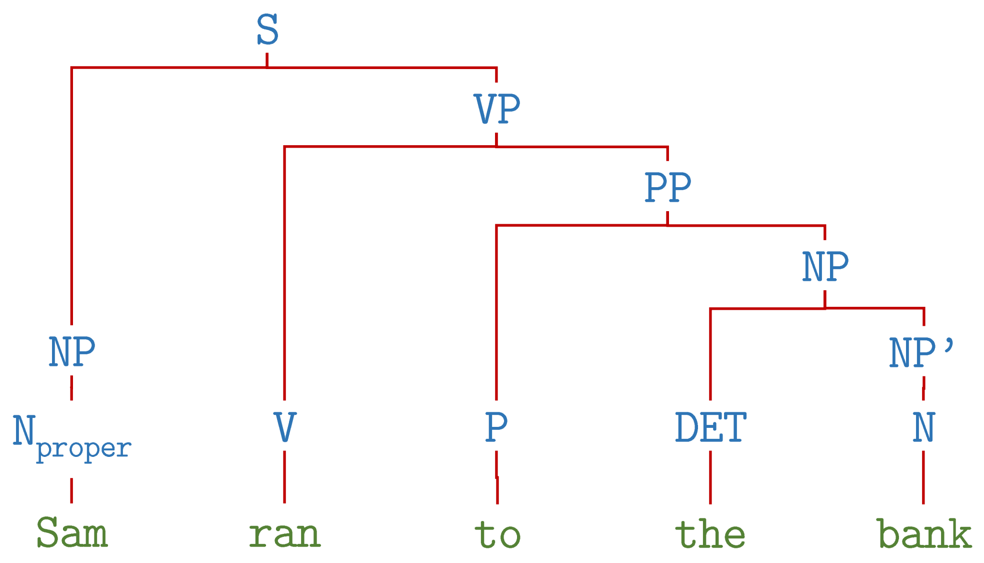
- Another way to think about grammars is as a deductive process.
- Each production rule can be interpreted as a deduction rule.
- E.g., from ‘the’, we can deduce DET from the rule DET -> ‘the’.
- From NP and VP, we can deduce S from the rule S -> NP VP.

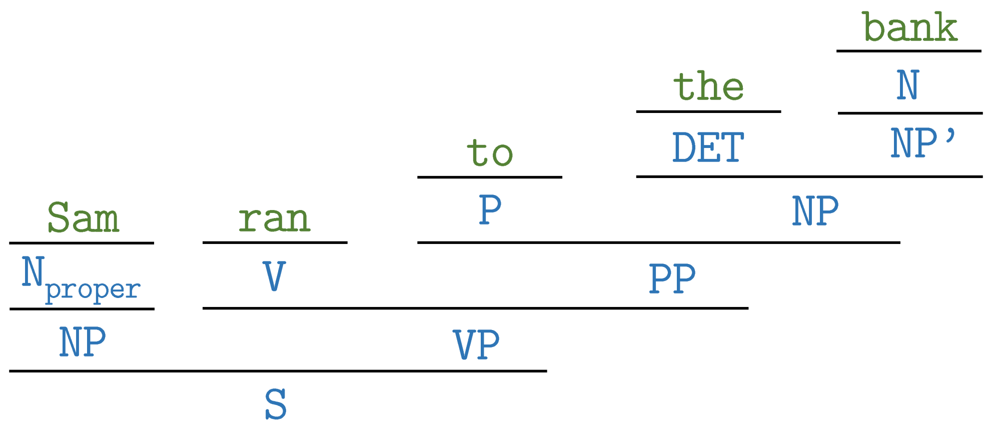




GRAMMAR

- The syntax tree is effectively a proof tree:
  - We have "proven" S from '`Sam ran to the bank`'.
- Thus, syntax trees are also interchangeably called derivation trees.
- The task of parsing is equivalent to deriving/proving S from an input string.





GRAMMAR

- It is easier to parse some grammars than others.
- Consider the language containing strings with only 1's.
  - It's easy to recognize strings in this language:
  - Just check if the input string contains only 1's.
- Can we write this as a grammar?
- Note the two grammars below generate the same set of strings.
  - Thus, they are called weakly equivalent.

```
S -> '1' S
S -> '1'
```

or

```
S -> S '1'
S -> '1'
```

CONTEXT-FREE GRAMMARS

- Context-free grammars are grammars where each production rule has form:

  `A -> α`

  where `A` is a nonterminal,

  and `α` is any sequence containing terminal or nonterminal symbols.

- A context-free language is a language generated by a context-free grammar.
  - E.g., the language of strings containing ‘`(`’ and ‘`)`’ where the parentheses are balanced.

    `L = {‘()’, ‘()()’, ‘(())’, ‘((()(()))())()’, …}`

CONTEXT-FREE GRAMMARS

- Context-free grammars are grammars where each production rule has form:

  `A -> α`

  where `A` is a nonterminal,

  and α is any sequence containing terminal or nonterminal symbols.
- A context-free language is a language generated by a context-free grammar.
  - E.g., the language of strings containing ‘(’ and ‘)’ where the parentheses are balanced.

```
S -> ‘(’ S ‘)’
S -> S S
S -> ‘()’
```

CONTEXT-FREE GRAMMARS

- Another example CFG is the fragment of English from the previous lecture:

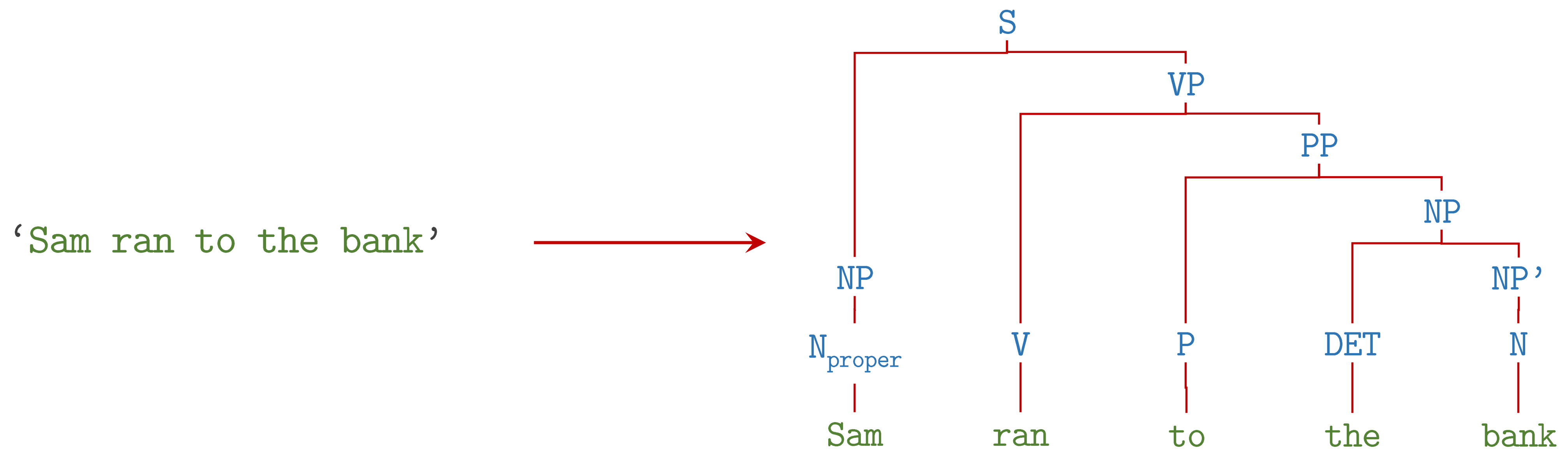
S -> NP VP
VP -> V NP
VP -> V PP
PP -> P NP
NP -> NP PP
NP -> N_{proper}
NP -> DET NP’
NP’ -> N

V -> ‘ran’
V -> ‘sees’
V -> ‘see’
P -> ‘to’
P -> ‘near’
N -> ‘bank’
N -> ‘coffee’
N -> ‘shop’

N_{proper} -> ‘Ada’
N_{proper} -> ‘Alex’
N_{proper} -> ‘Sam’
N_{proper} -> ‘Zach’
DET -> ‘the’
DET -> ‘a’

PARSING CFGS

- (Syntactic) parsing is the task of converting a sentence into a syntax tree, given a grammar.



CHOMSKY NORMAL FORM

- Some CFGs are written in a simpler form called Chomsky normal form (Chomsky 1959).
- A CFG is in Chomsky normal form if all of its production rules are of the form:

```
A -> B C
A -> ‘t’
```

  where `A`, `B`, and `C` are nonterminals and `t` is a terminal.

- Some parsing algorithms are simpler to describe when applied to grammars in Chomsky normal form.
- Any CFG can be converted into an equivalent CFG in Chomsky normal form.

CONVERTING TO CHOMSKY NORMAL FORM

- Take the balanced-parentheses grammar from earlier:

```
S -> ‘(’ S ‘)’
S -> S S
S -> ‘()’
```

- The first step is to change any production rule with mixed terminal-nonterminal symbols in the right-hand side into an equivalent rule that contains only nonterminals.

$S \rightarrow P_L\ S\ P_R$ $\qquad$ $P_L \rightarrow$ ‘(’

$S \rightarrow S\ S$ $\qquad$ $P_R \rightarrow$ ‘)’

$S \rightarrow$ ‘()’

CONVERTING TO CHOMSKY NORMAL FORM

- Next, repeat the following:
  - For any rule with more than two symbols in the right-hand side,
    - E.g., $A \rightarrow B_1\ B_2\ \dots\ B_n$
  - Create a new nonterminal that replaces the last n – 1 symbols,
    - E.g., $A \rightarrow B_1\ C$
  - And create a new production rule for this replacement.
    - E.g., $C \rightarrow B_2\ \dots\ B_n$

$S \rightarrow P_L\ S\ P_R$

$S \rightarrow S\ S$

$S \rightarrow$ ‘()’

$P_L \rightarrow$ ‘(’

$P_R \rightarrow$ ‘)’

CONVERTING TO CHOMSKY NORMAL FORM

- Next, repeat the following:
  - For any rule with more than two symbols in the right-hand side,
    - E.g., `A -> `B_1 B_2 … B_n
  - Create a new nonterminal that replaces the last n – 1 symbols,
    - E.g., `A -> `B_1 `C`
  - And create a new production rule for this replacement.
    - E.g., `C -> `B_2 … B_n

S -> P_L S_2	P_L -> '('
S -> S S	P_R -> ')'
S -> '()'	S_2 -> S P_R

CONVERTING TO CHOMSKY NORMAL FORM

- If there are any "unit" rules `A -> B`, for nonterminals `A` and `B`,
  - Convert any rule of the form `B -> ...` into `A -> ...`
  - And we can remove the rule `A -> B` as well.
- This type of rule doesn't appear in the example grammar below.
- But may appear in CFGs more generally.

$S \rightarrow P_L\ S_2$	$P_L \rightarrow$ '('
$S \rightarrow S\ S$	$P_R \rightarrow$ ')'
$S \rightarrow$ '()'	$S_2 \rightarrow S\ P_R$

PARSING CFGS

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- The algorithm was described by Kasami (1965) and Younger (1967), and rediscovered later by Cocke and Schwartz (1970).

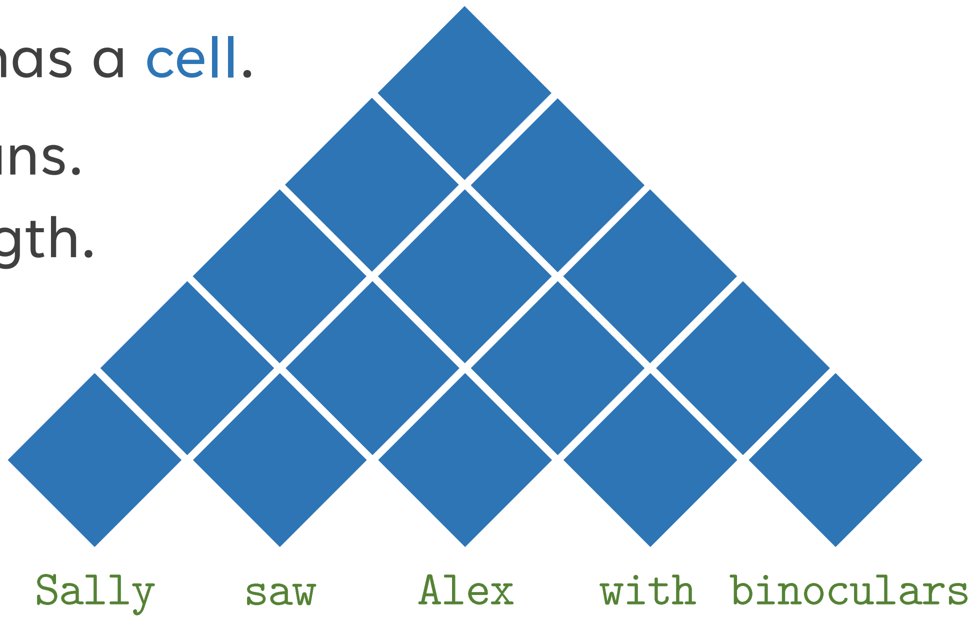
PARSING CFGS

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- In dynamic programming, we solve a problem by breaking it down into simpler subproblems.
  - The solutions to the subproblems makes it easier to solve the full problem.
- In parsing CFGs,
  - Suppose we are given a sentence '`Sally saw Alex with binoculars.`'
  - And suppose we know that '`Sally`' is a noun phrase (`NP`) and '`saw Alex with binoculars`' is a verb phrase (`VP`).
  - Our grammar contains the rule `S -> NP VP`.
  - We can conclude that the full sentence can be parsed as `S`.

CKY PARSING

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- We use a data structure called a chart.
    - For each span `(i,j)` the chart has a cell.
- Start parsing with the smallest spans.
    - Progressively increase span length.

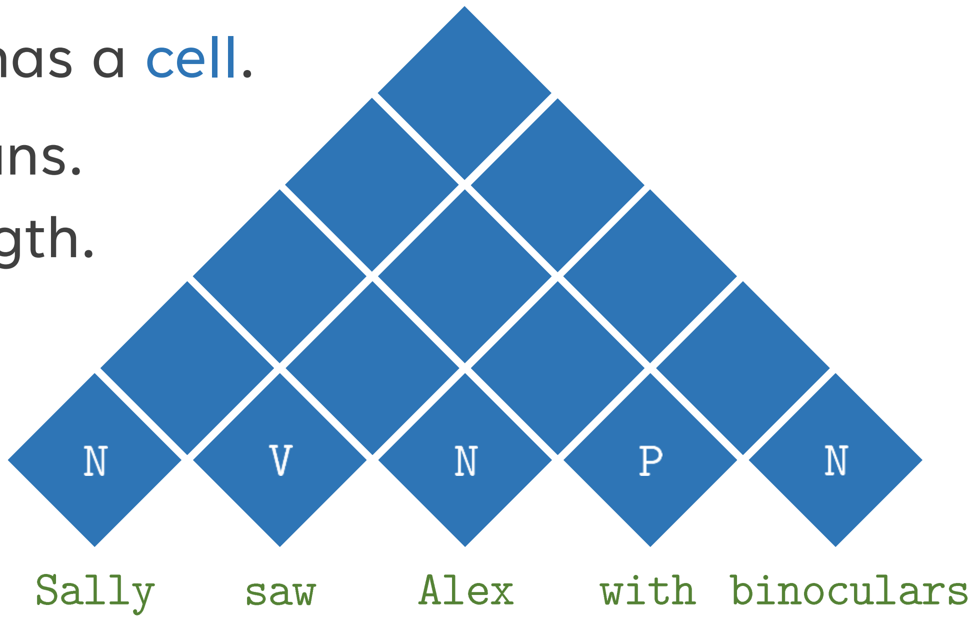
```
S  -> NP VP      V -> 'saw'
VP ->  V NP      P -> 'with'
VP -> VP PP      N -> 'binoculars'
PP ->  P NP      N -> 'Sally'
NP -> NP PP      N -> 'Alex'
NP ->  N
```



CKY PARSING

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- We use a data structure called a chart.
    - For each span `(i,j)` the chart has a cell.
- Start parsing with the smallest spans.
    - Progressively increase span length.

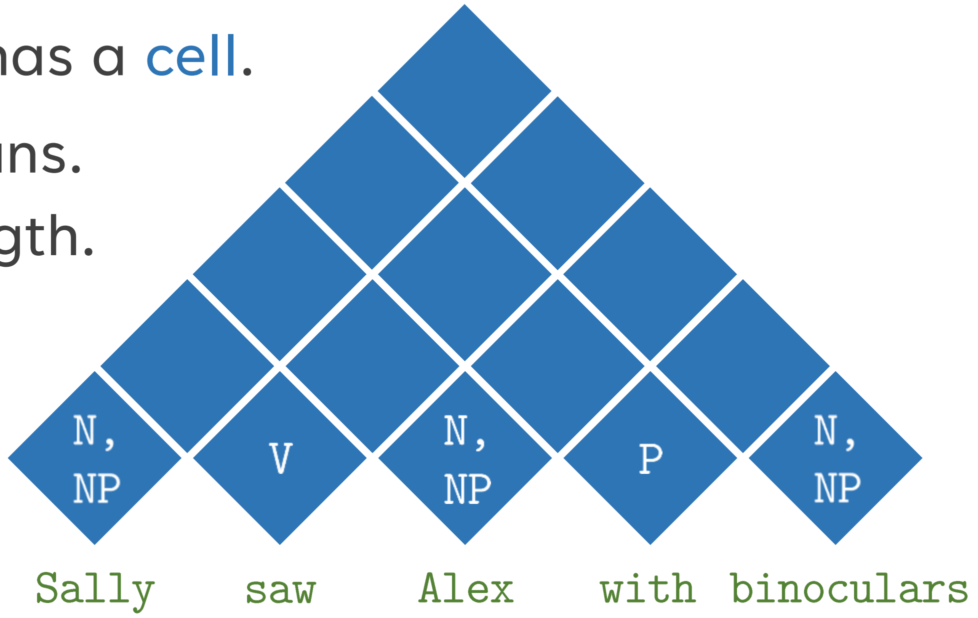
```
S  -> NP VP        V -> 'saw'
VP ->  V NP        P -> 'with'
VP -> VP PP        N -> 'binoculars'
PP ->  P NP        N -> 'Sally'
NP -> NP PP        N -> 'Alex'
NP ->  N
```



CKY PARSING

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- We use a data structure called a chart.
  - For each span `(i,j)` the chart has a cell.
- Start parsing with the smallest spans.
  - Progressively increase span length.

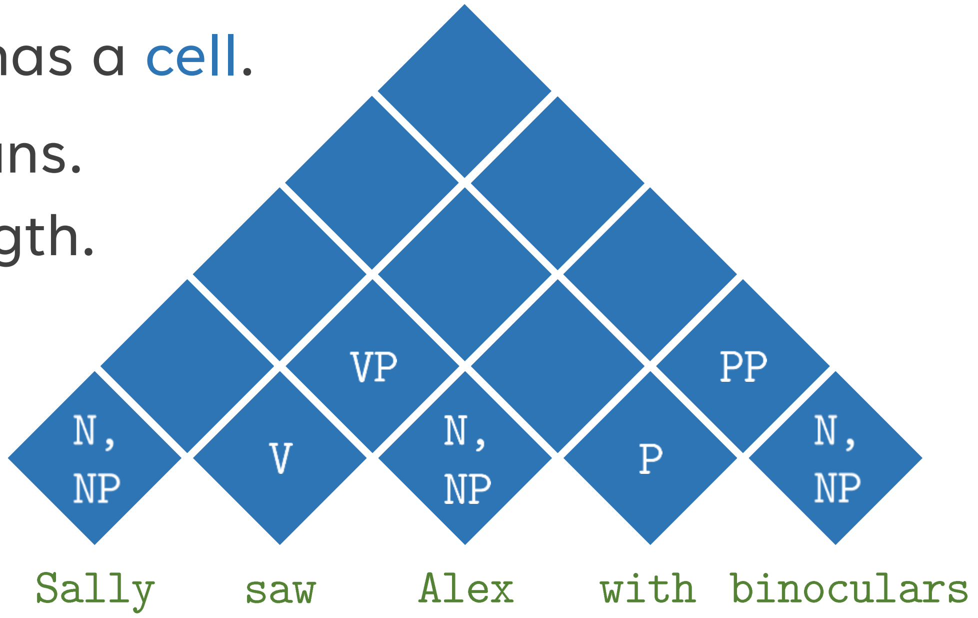
```
S  -> NP VP       V -> 'saw'
VP ->  V NP       P -> 'with'
VP -> VP PP       N -> 'binoculars'
PP ->  P NP       N -> 'Sally'
NP -> NP PP       N -> 'Alex'
NP ->  N
```



CKY PARSING

- The CKY (Cocke-Kasami-Younger; or CYK) algorithm is a simple application of dynamic programming to parsing CFGs in Chomsky normal form.
- We use a data structure called a chart.
  - For each span `(i,j)` the chart has a cell.
- Start parsing with the smallest spans.
  - Progressively increase span length.

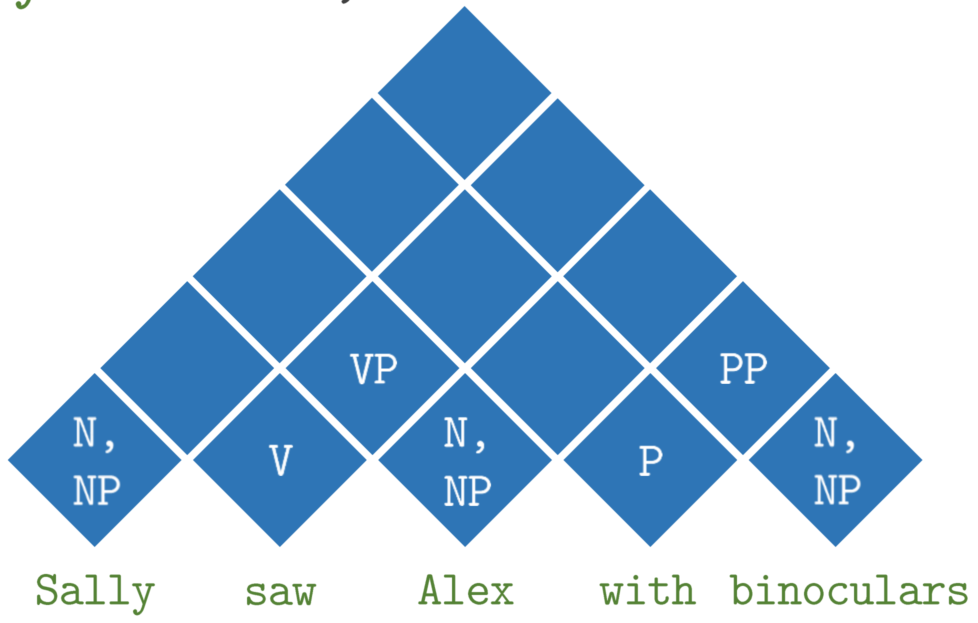
```
S  -> NP VP       V -> 'saw'
VP ->  V NP       P -> 'with'
VP -> VP PP       N -> 'binoculars'
PP ->  P NP       N -> 'Sally'
NP -> NP PP       N -> 'Alex'
NP ->  N
```



CKY PARSING

- For each cell `(i,j)`, we have to consider all pairs of cells `(i,k),(k,j)` for all `k` such that `i` $\leq$ `k` $\leq$ `j`.
- For example, for the cell with '`Sally saw Alex,`'

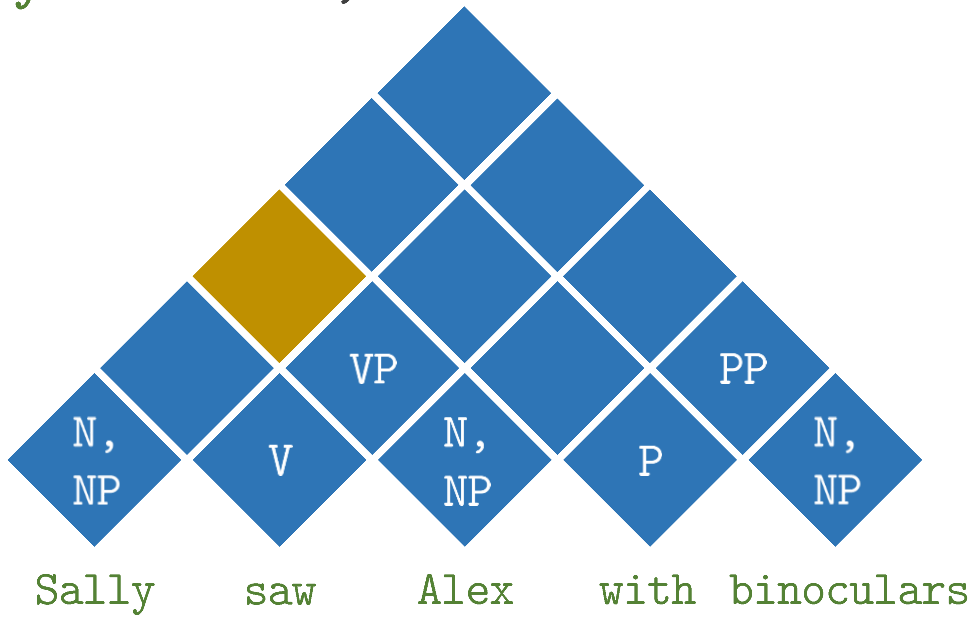
```
S  -> NP VP       V -> 'saw'
VP ->  V NP       P -> 'with'
VP -> VP PP       N -> 'binoculars'
PP ->  P NP       N -> 'Sally'
NP -> NP PP       N -> 'Alex'
NP ->  N
```



CKY PARSING

- For each cell `(i,j)`, we have to consider all pairs of cells `(i,k),(k,j)` for all `k` such that `i ≤ k ≤ j`.
- For example, for the cell with '`Sally saw Alex,`'
- We must consider combining both:

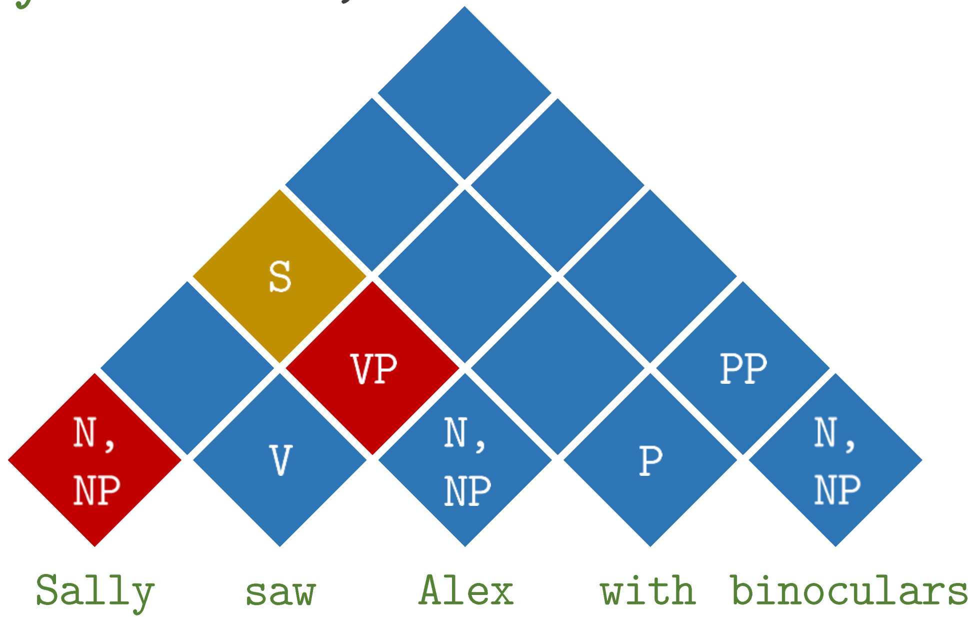
```
S  -> NP VP      V -> 'saw'
VP ->  V NP      P -> 'with'
VP -> VP PP      N -> 'binoculars'
PP ->  P NP      N -> 'Sally'
NP -> NP PP      N -> 'Alex'
NP ->  N
```



CKY PARSING

- For each cell (i,j), we have to consider all pairs of cells (i,k),(k,j) for all k such that i ≤ k ≤ j.
- For example, for the cell with 'Sally saw Alex,'
- We must consider combining both:
    - 'Sally' and 'saw Alex'

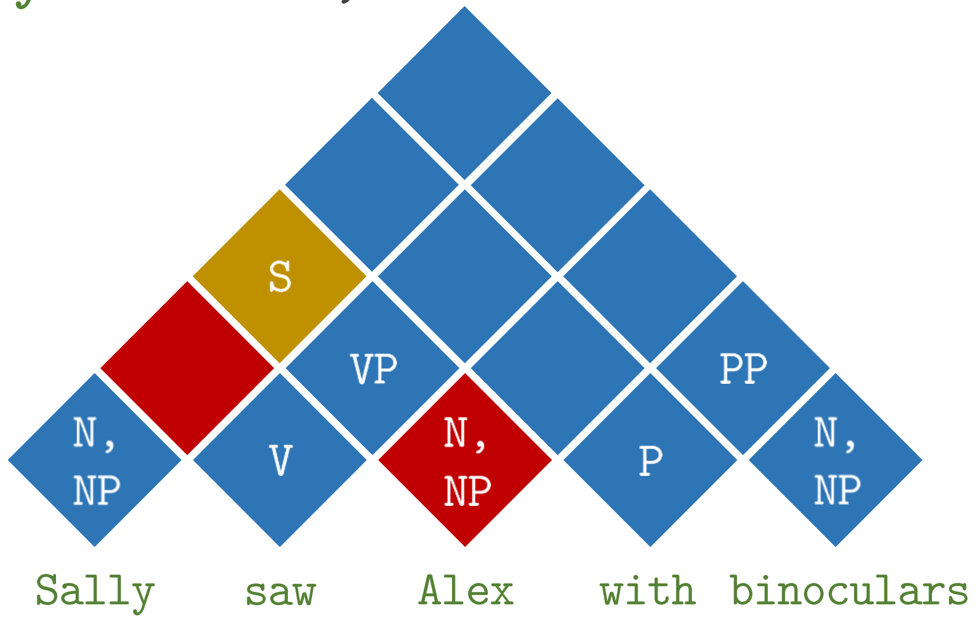
```
S  -> NP VP      V -> 'saw'
VP ->  V NP      P -> 'with'
VP -> VP PP      N -> 'binoculars'
PP ->  P NP      N -> 'Sally'
NP -> NP PP      N -> 'Alex'
NP ->  N
```



CKY PARSING

- For each cell `(i,j)`, we have to consider all pairs of cells `(i,k),(k,j)` for all `k` such that `i ≤ k ≤ j`.
- For example, for the cell with '`Sally saw Alex,`'
- We must consider combining both:
  - '`Sally`' and '`saw Alex`'
  - '`Sally saw`' and '`Alex`'

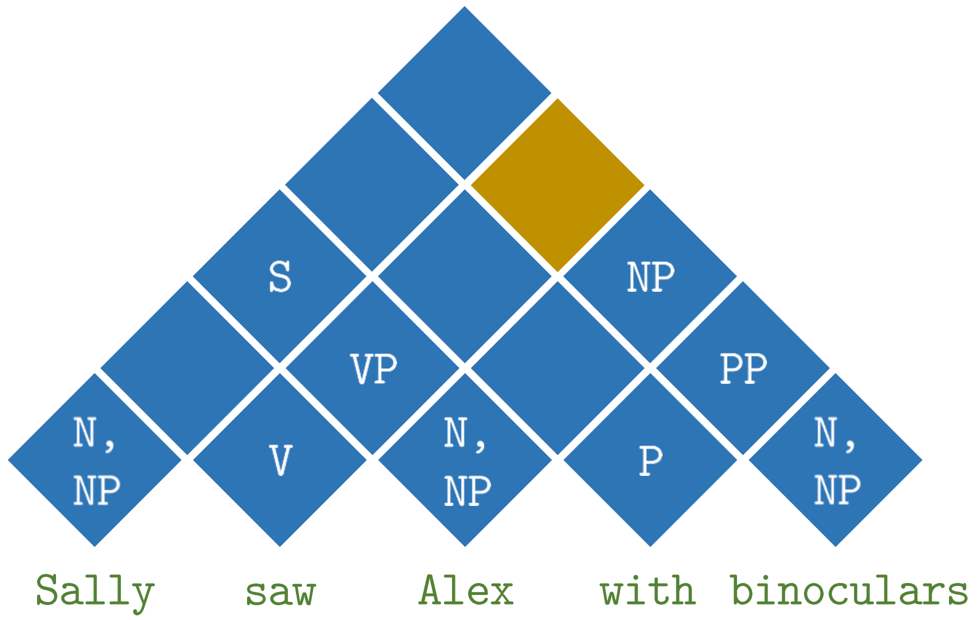
```
S  -> NP VP      V -> 'saw'
VP ->  V NP      P -> 'with'
VP -> VP PP      N -> 'binoculars'
PP ->  P NP      N -> 'Sally'
NP -> NP PP      N -> 'Alex'
NP ->  N
```



CKY PARSING

- Similarly for the span '`saw Alex with binoculars`' we must consider:

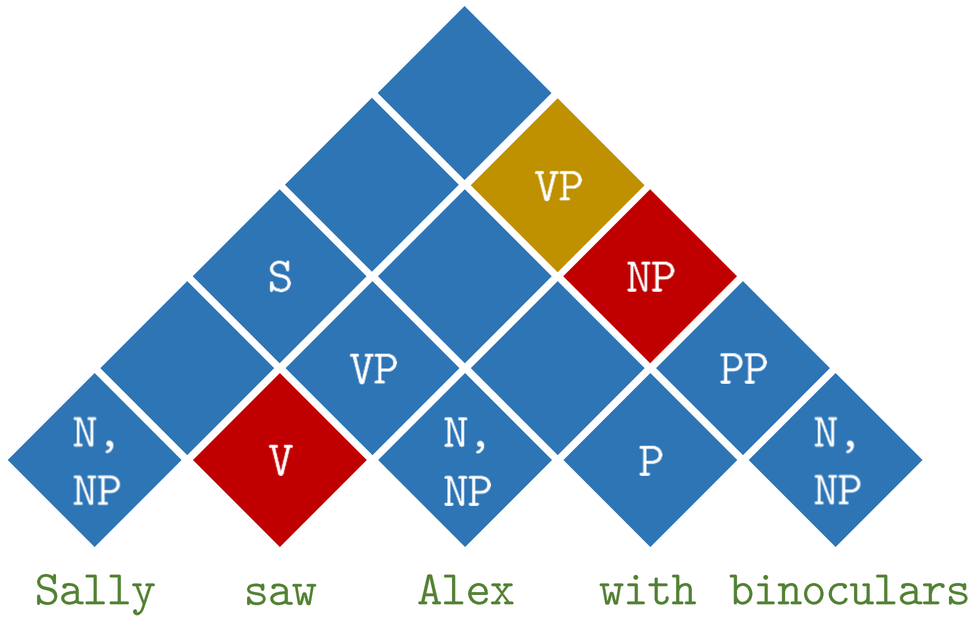
```
S  -> NP VP        V -> 'saw'
VP ->  V NP        P -> 'with'
VP -> VP PP        N -> 'binoculars'
PP ->  P NP        N -> 'Sally'
NP -> NP PP        N -> 'Alex'
NP ->  N
```



CKY PARSING

- Similarly for the span ‘saw Alex with binoculars’ we must consider:
  - ‘saw’ and ‘Alex with binoculars’

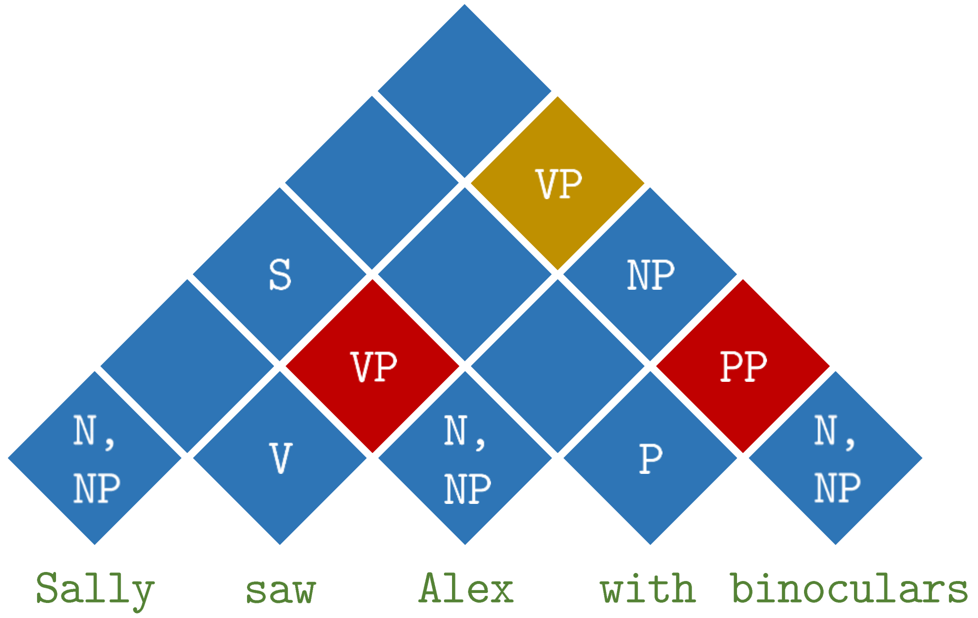
```
S  -> NP VP       V -> ‘saw’
VP ->  V NP       P -> ‘with’
VP -> VP PP       N -> ‘binoculars’
PP ->  P NP       N -> ‘Sally’
NP -> NP PP       N -> ‘Alex’
NP ->  N
```



CKY PARSING

- Similarly for the span ‘saw Alex with binoculars’ we must consider:
  - ‘saw’ and ‘Alex with binoculars’
  - ‘saw Alex’ and ‘with binoculars’

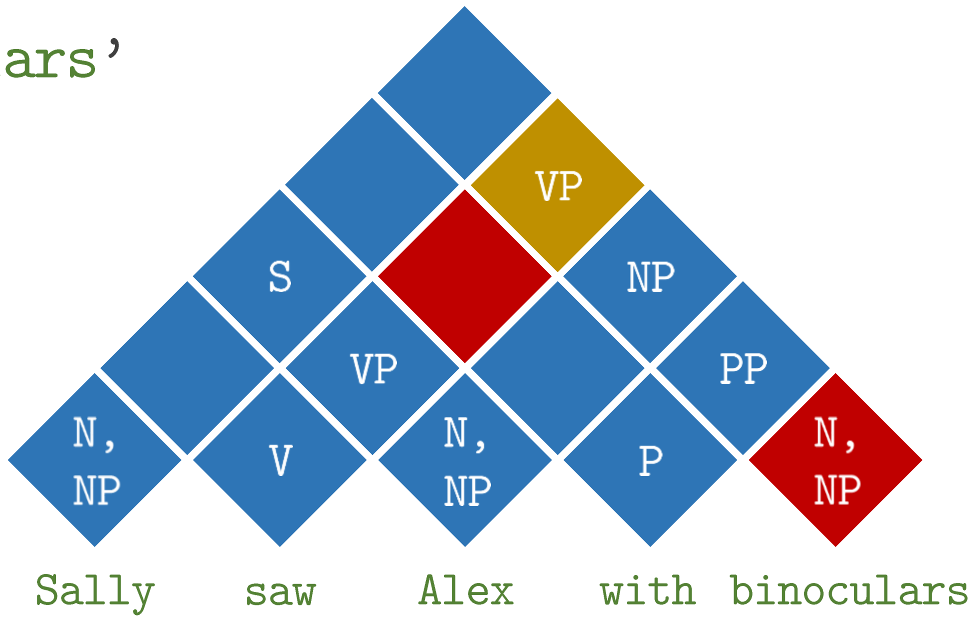
```
S  -> NP VP        V -> 'saw'
VP ->  V NP        P -> 'with'
VP -> VP PP        N -> 'binoculars'
PP ->  P NP        N -> 'Sally'
NP -> NP PP        N -> 'Alex'
NP ->  N
```



CKY PARSING

- Similarly for the span '`saw Alex with binoculars`' we must consider:
  - '`saw`' and '`Alex with binoculars`'
  - '`saw Alex`' and '`with binoculars`'
  - '`saw Alex with`' and '`binoculars`'

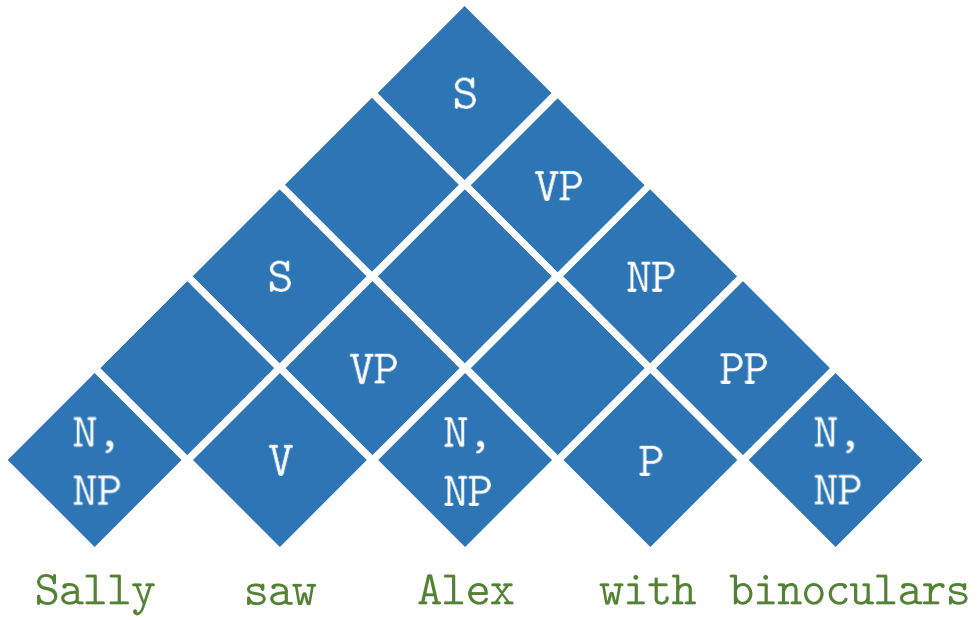
```
S  -> NP VP        V -> 'saw'
VP ->  V NP        P -> 'with'
VP -> VP PP        N -> 'binoculars'
PP ->  P NP        N -> 'Sally'
NP -> NP PP        N -> 'Alex'
NP ->  N
```



CKY PARSING

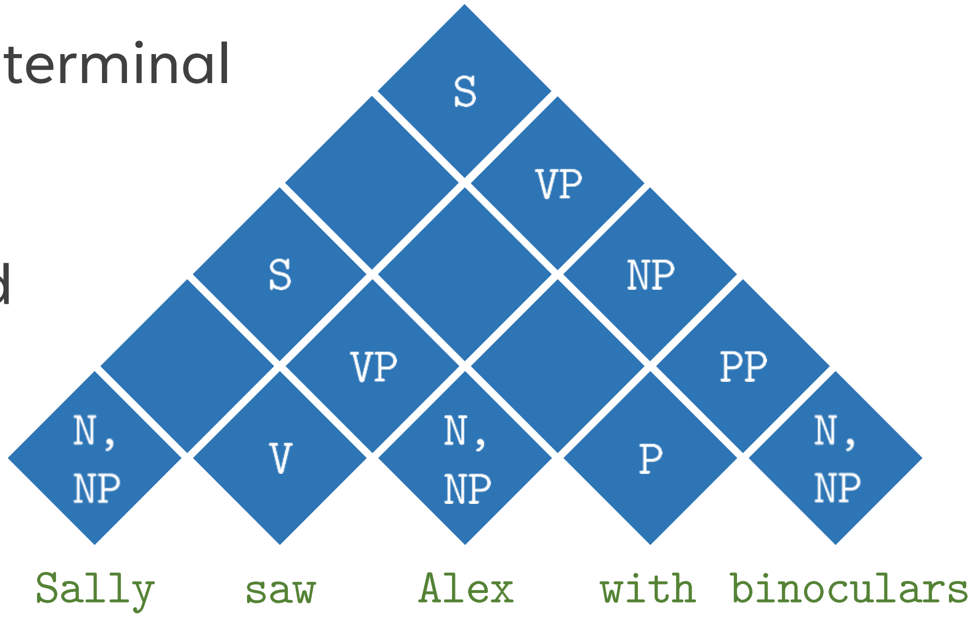
- We continue until we have processed all cells in the chart.
- In its simplest form, each cell simply records which nonterminals were parsed for that corresponding span.

```
S  -> NP VP       V -> 'saw'
VP ->  V NP       P -> 'with'
VP -> VP PP       N -> 'binoculars'
PP ->  P NP       N -> 'Sally'
NP -> NP PP       N -> 'Alex'
NP ->  N
```



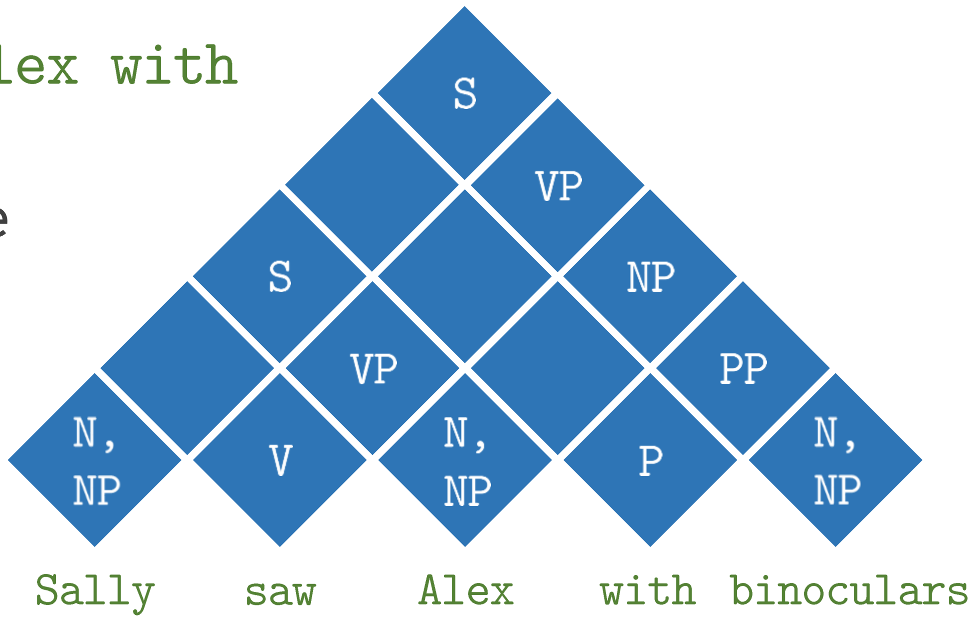
CKY PARSING

- We continue until we have processed all cells in the chart.
- In its simplest form, each cell simply records which nonterminals were parsed for that corresponding span.
- If instead, we record *how* each nonterminal was constructed in each cell,
- E.g., the `VP` for the span '`saw Alex with binoculars`' was constructed from the `V` '`saw`' and the `NP` '`Alex with binoculars,`'
- We can reconstruct the syntax tree by following the pointers from the root of the chart.



CKY PARSING

- CKY can also parse ambiguous sentences.
- Here, the VP for the span ‘saw Alex with binoculars’ can be constructed in two ways:
  1. from the V ‘saw’ and the NP ‘Alex with binoculars,’
  2. from the VP ‘saw Alex’ and the PP ‘with binoculars.’
- Both these constructions must be stored in the cell for ‘saw Alex with binoculars.’



QUESTIONS?