NUS | Computing
National University of Singapore

CS4248: Natural Language Processing

Lecture 9 — Trees

Announcements

Outline

- **Syntactic Parsing**
  - **Quick recap: RegEx**
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- CYK Parsing Algorithm
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

Throwback — Regular Expression

- Equivalence
  - Regular Expressions describe **Regular Languages**
    (most restricted types of languages w.r.t Chomsky Hierarchy)
  - Regular Language = language accepted by a FSA

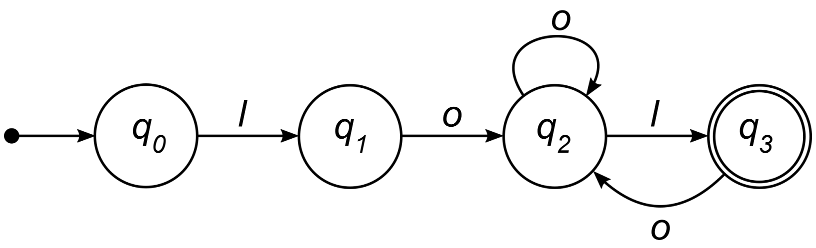
Regular Expression

l(o+l)+

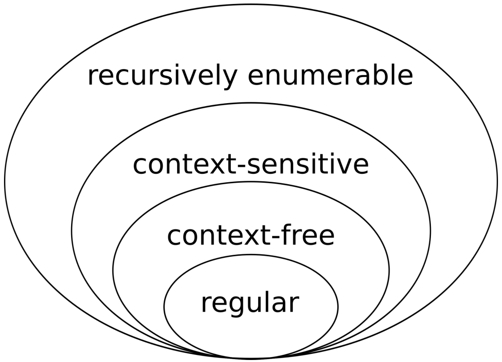
Regular Language

{lol, loool, lolol, looolol, …}

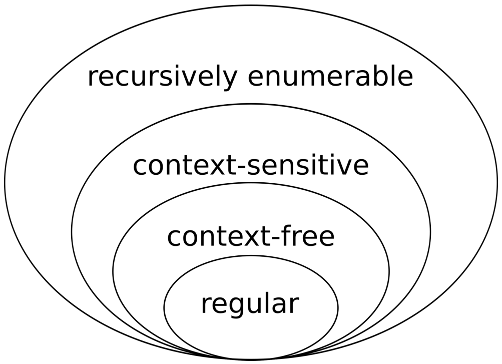
Example: FSA that accepts the Regular Language described by the Regular Expression **l(o+l)+**



Chomsky Hierarchy
(Source: Wikipedia)



Regular Expressions — Limitations



- Not all languages can be described using RegEx
  - Example:

$$\{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, 00001111, \ldots\}$$

➜ Natural Language is not a Regular Language

- Natural language allows for nested structures (**center embeddings**)

The food was delicious

The food Alice cooked was delicious

The food Alice the sister of Bob cooked was delicious

Quick Quiz

Syntax & Constituency

- Important questions
  - How are words combined to form phrases?
  - How are phrases combined to form longer phrases?
  - How are phrases are combined to form sentences?

How meaning is mapped onto what language structures?

- Important concept: constituency = phrase structure
  - Constituent = group of words that behave as a single unit

Constituents

- Constituent — Definition
  - Group of words that behaves as a single unit or phrase
    (by default: individual words are constituents, but there are exceptions)
  - Sentences can be described as a hierarchical structure of constituents (in a bit: parse trees)

- Question: How do we know a group of words forms a constituent?
  - Handwavy answer: Group of words "makes sense" on its own

"She heard a loud shot from the bank during the time of the robbery."

a loud shot

a loud shot from the

a loud shot from the bank

  - Formal answer: **Constituency Tests**

Constituency Tests (some examples)

- Topicalization
  - Only a constituent can be moved to different locations in a sentence

*"They met **at 8 pm** for lunch." — "They met for lunch **at 8 pm**." — "**At 8 pm**, they met for lunch."*

- Proform substitution
  - Only a constituent can be substituted with a proform like *it*, *that*, *them*, *then*, *there*, etc.

*"Chris went **back to Germany**."* ➜ *"Chris went <u>there</u>."*

- Fragment Answers
  - Only a constituent can answer a question, while retaining the meaning of the original sentence.

*"Alice was hit by **the green car**." — Q: "What hit Alice?"* ➜ *"The green car."*

Quick Quiz

Outline

- **Syntactic Parsing**
  - Quick recap: RegEx
  - **Context-free grammars**
  - Structural Ambiguity
  - Chomsky Normal Form

- CYK Parsing Algorithm
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

Context-Free Grammars (CFGs)

- Context-Free Grammars
  - Most common way to capture constituency and ordering ➜ good fit for natural language!
    (in fact, context-free grammars were first used to study human languages to describe the structure of sentences)
  - Define what meaningful constituents are and how a constituent is formed out of other constituents
  - More powerful than RegExs as they can express recursive structure
    (in contrast, context free grammars can describe regular languages)

- Example

special start symbol

$$\underbrace{\begin{aligned} \mathrm{S} &\rightarrow \mathrm{NP\ VP} \\ \mathrm{NP} &\rightarrow \mathrm{Det\ Noun} \\ \mathrm{VP} &\rightarrow \mathrm{Verb\ NP} \\ \mathrm{Det} &\rightarrow a \mid the \\ \mathrm{Noun} &\rightarrow man \mid meal \mid flight \\ \mathrm{Verb} &\rightarrow saw \mid booked \end{aligned}}_{\text{set of rules or productions}}$$

set of **rules** or **productions**

Non-terminal symbols
- Symbols that can be replaced according to rules
- For natural language grammars: phrase names, part of speech

Terminal symbols
- May be the output of a rule; cannot be changed/replaced further
- For natural language grammars: words/tokens

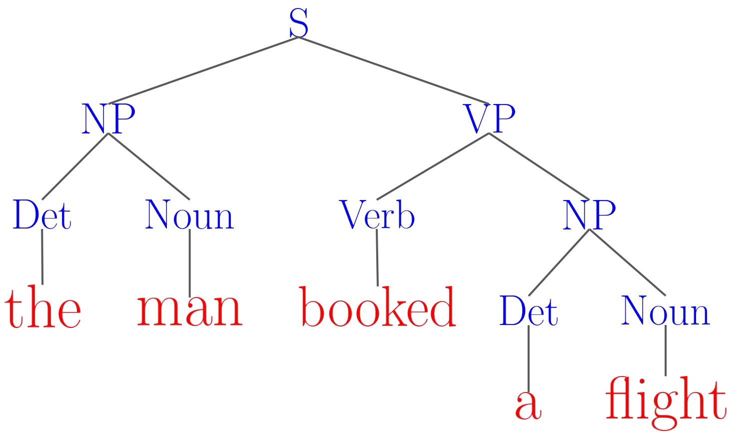
Context-Free Grammars (CFGs)

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
NP &\rightarrow Det\ Noun \\
VP &\rightarrow Verb\ NP \\
Det &\rightarrow a \mid the \\
Noun &\rightarrow man \mid meal \mid flight \\
Verb &\rightarrow saw \mid booked
\end{aligned}
$$

- Application of rules — example

$$
\begin{aligned}
S &\rightarrow \mathrm{NP\ VP} \\
&\rightarrow \mathrm{Det\ Noun\ VP} \\
&\rightarrow \textit{the}\ \mathrm{Noun\ VP} \\
&\rightarrow \textit{the man}\ \mathrm{VP} \\
&\rightarrow \textit{the man}\ \mathrm{Verb\ NP} \\
&\rightarrow \textit{the man booked}\ \mathrm{NP} \\
&\rightarrow \textit{the man booked}\ \mathrm{Det\ Noun} \\
&\rightarrow \textit{the man booked a}\ \mathrm{Noun} \\
&\rightarrow \textit{the man booked a flight}
\end{aligned}
$$

Visualization as **Parse Tree**



Derivation: sequence of rules/productions used to generate a string of words

CFG — Formal Definition

- A CFG is a 4 tuple $\langle N, \Sigma, R, S \rangle$
  - N — set of non-terminal symbols
  - $\sum$ — set of terminal symbols
  - R — set of rules

Allowed format for all rules

$A \rightarrow \alpha$ with $A \in N,\ \alpha \in (N \cup \Sigma)$

  - S — start symbol

These restrictions make a grammar context-free!

Example

$N = \{\mathrm{NP},\ \mathrm{VP},\ \mathrm{Det},\ \mathrm{Noun},\ \mathrm{Verb}\}$

$\Sigma = \{a,\ the,\ man,\ meal,\ flight,\ saw,\ booked\}$

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{NP\ VP} \\ \mathrm{NP} &\rightarrow \mathrm{Det\ Noun} \\ \mathrm{VP} &\rightarrow \mathrm{Verb\ NP} \\ \mathrm{Det} &\rightarrow a \mid the \\ &\ldots \end{aligned}$$

CFG — Running Example

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{S} &\rightarrow \text{Aux NP VP} \\
\text{S} &\rightarrow \text{VP} \\
\text{NP} &\rightarrow \text{Pronoun} \\
\text{NP} &\rightarrow \text{ProperNoun} \\
\text{NP} &\rightarrow \text{Det Nominal} \\
\text{Nominal} &\rightarrow \text{Noun} \\
\text{Nominal} &\rightarrow \text{Nominal Noun} \\
\text{Nominal} &\rightarrow \text{Nominal PP} \\
\text{VP} &\rightarrow \text{Verb} \\
\text{VP} &\rightarrow \text{Verb NP} \\
\text{VP} &\rightarrow \text{Verb NP PP} \\
\text{VP} &\rightarrow \text{Verb PP} \\
\text{VP} &\rightarrow \text{VP PP} \\
\text{PP} &\rightarrow \text{Prep NP}
\end{aligned}
$$

$$
\begin{aligned}
\text{Det} &\rightarrow the \mid a \mid that \mid this \\
\text{Noun} &\rightarrow book \mid flight \mid meal \mid money \\
\text{Verb} &\rightarrow book \mid include \mid prefer \\
\text{Pronoun} &\rightarrow I \mid she \mid he \mid me \\
\text{ProperNoun} &\rightarrow Singapore \mid Frankfurt \mid SIA \\
\text{Aux} &\rightarrow do \mid does \mid did \\
\text{Prep} &\rightarrow from \mid to \mid on \mid near \mid through
\end{aligned}
$$

Important requirements to make it a CFG

- Only single non terminals on the left-hand side

➜ Application of a rule does not depend on a context

Quick Quiz

Outline

- **Syntactic Parsing**
  - Quick recap: RegEx
  - Context-free grammars
  - **Structural Ambiguity**
  - Chomsky Normal Form
- CYK Parsing Algorithm
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

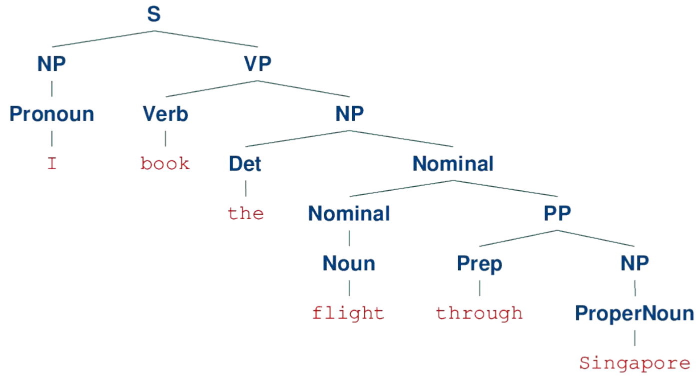
Ambiguity

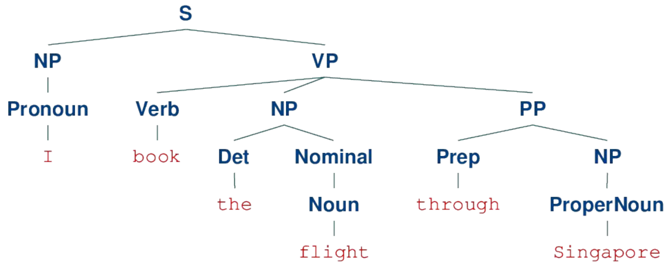
- Ambiguity of Natural Language
  - Common: multiple ways to interpret a sentence
  - Different interpretation ➜ different meaning

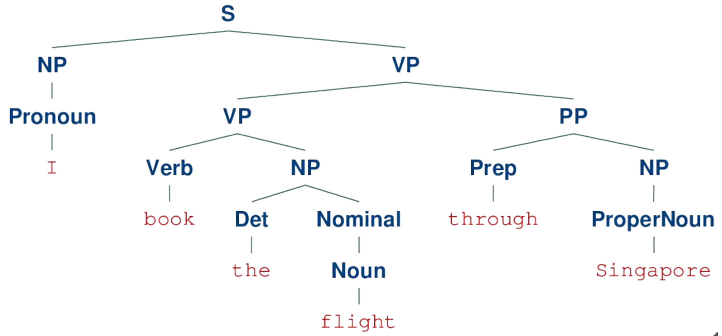
➜ Structural Ambiguity

- A grammar can assign more than one parse to a sentence
- Example (using our toy grammar):

  "I book the flight through Singapore"







Structural Ambiguity

- Two common types of Structural Ambiguity

(1) Attachment Ambiguity

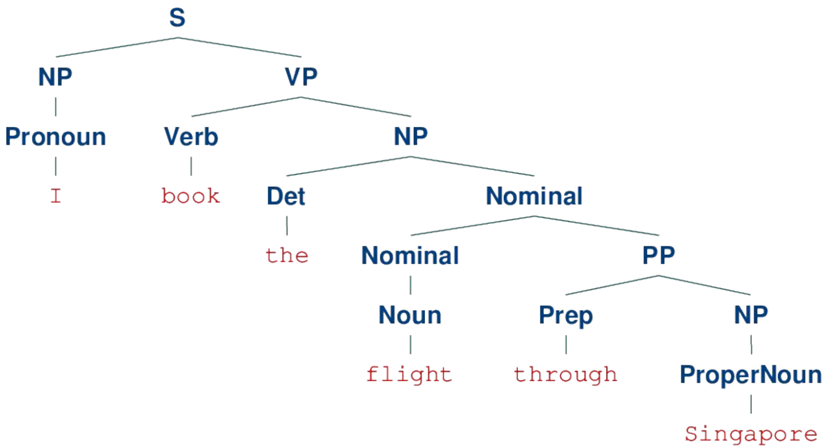
- A particular constituent can be attached to the parse tree at more than one place

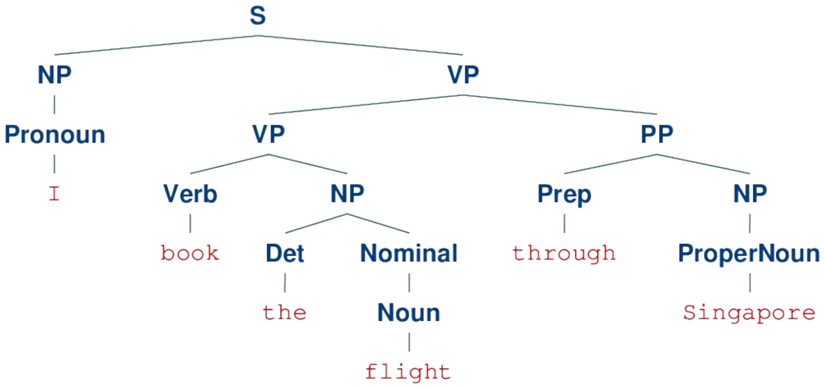
(2) Coordination Ambiguity

- Phrases can be conjoined by conjunction like *"and"*, *"or"*, *"but"*, *"because"*, *"if"*, etc.
- Different types of conjunctions
  (coordinating conjunctions, correlative conjunctions, correlative conjunctions)

Attachment Ambiguity

"I book the flight through Singapore"





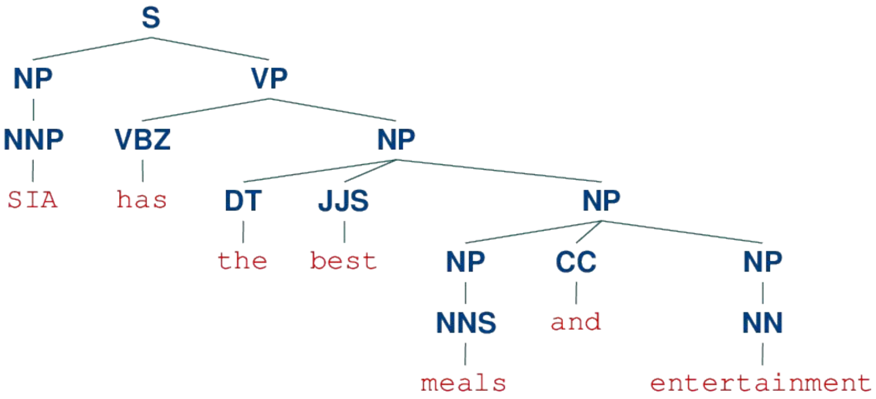
"through Singapore" attached to noun phrase related to *"flight"*

"through Singapore" attached to verb phrase related to *"book"*

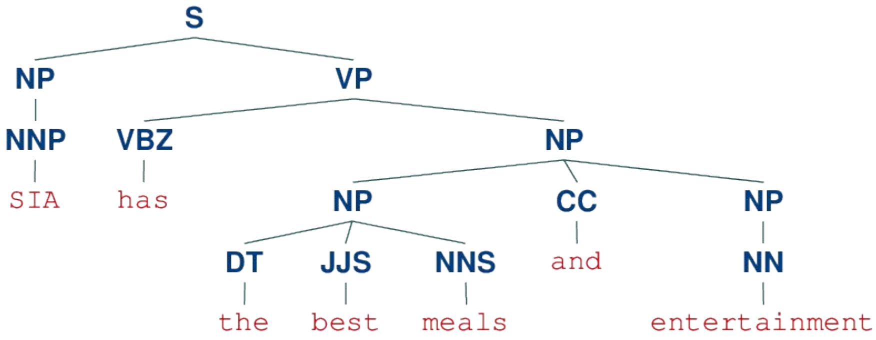
(like booking a flight through an agent)

Coordination Ambiguity

"SIA has the best meals and entertainment"



"best" refers to both the meals and the entertainment



"best" refers to only the meals but not the entertainment

Note: This example used a different grammar since our toy grammar does not support conjunctions.

Summary So Far...

- Parsing as a 2-part task

  (1) Syntactic Parsing

  - Extract all possible parses for a sentence
  - Typically requires a grammar transformation step
    ("binarization" of grammar to ensure efficient parsing)

  (2) Syntactic Disambiguation

  - Score all parses and return the best parse
  - Scores commonly expressed as probability

Outline

- **Syntactic Parsing**
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - **Chomsky Normal Form**
- CYK Parsing Algorithm
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

Grammar Transformation (for CFGs)

- Important requirement: binarized rules
  - No more than 2 non-terminals on the right-hand side of rules
  - Crucial for a cubic time parsing of CFGs

➜ Common transformation: **Chomsky Normal Form**

- Restrictions on rules compared to general CFG

Allowed format for all rules

$A \rightarrow \alpha$ with $A \in N, \quad \alpha \in (N \cup \Sigma)$

α is either 1 terminal OR 2 non-terminals

Grammar Transformation

Allowed format for all rules

$A \rightarrow \alpha$ with $A \in N, \quad \alpha \in (N \cup \Sigma)$

α **is either 1 terminal OR 2 non-terminals**

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{S} &\rightarrow \text{Aux NP VP} \\
\text{S} &\rightarrow \text{VP} \\
\text{NP} &\rightarrow \text{Pronoun} \\
\text{NP} &\rightarrow \text{ProperNoun} \\
\text{NP} &\rightarrow \text{Det Nominal} \\
\text{Nominal} &\rightarrow \text{Noun} \\
\text{Nominal} &\rightarrow \text{Nominal Noun} \\
\text{Nominal} &\rightarrow \text{Nominal PP} \\
\text{VP} &\rightarrow \text{Verb} \\
\text{VP} &\rightarrow \text{Verb NP} \\
\text{VP} &\rightarrow \text{Verb NP PP} \\
\text{VP} &\rightarrow \text{Verb PP} \\
\text{VP} &\rightarrow \text{VP PP} \\
\text{PP} &\rightarrow \text{Prep NP}
\end{aligned}
$$

$$
\begin{aligned}
\text{Det} &\rightarrow the \mid a \mid that \mid this \\
\text{Noun} &\rightarrow book \mid flight \mid meal \mid money \\
\text{Verb} &\rightarrow book \mid include \mid prefer \\
\text{Pronoun} &\rightarrow I \mid she \mid he \mid me \\
\text{ProperNoun} &\rightarrow Singapore \mid Frankfurt \mid SIA \\
\text{Aux} &\rightarrow do \mid does \mid did \\
\text{Prep} &\rightarrow from \mid to \mid on \mid near \mid through
\end{aligned}
$$

Chomsky Normal Form (CNF)

- Two basic transformation steps

(1) Recursive removal of unary rules (and empty rules)

$$\text{Nominal} \rightarrow \text{Noun} \qquad \text{Noun} \rightarrow book \mid flight \mid meal \mid money$$

➜

$$\text{Nominal} \rightarrow book \mid flight \mid meal \mid money \qquad \text{Noun} \rightarrow book \mid flight \mid meal \mid money$$

(2) Dividing n-ary rules by introducing new non-terminals

(n-ary rule = rule with n > 2 non-terminal on the right-hand side)

$$\text{S} \rightarrow \text{Aux NP VP}$$

➜

$$S \rightarrow X\ VP \qquad X \rightarrow Aux\ NP$$

Toy Grammar in Chomsky Normal Form CNF

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{S} &\rightarrow \text{X1 VP} \\
\text{X1} &\rightarrow \text{Aux NP} \\
\text{S} &\rightarrow \text{Verb NP} \\
\text{S} &\rightarrow \text{X2 PP} \\
\text{S} &\rightarrow \text{Verb PP} \\
\text{S} &\rightarrow \text{VP PP} \\
\text{NP} &\rightarrow \text{Det Nominal} \\
\text{Nominal} &\rightarrow \text{Nominal Noun} \\
\text{Nominal} &\rightarrow \text{Nominal PP} \\
\text{VP} &\rightarrow \text{Verb NP} \\
\text{VP} &\rightarrow \text{X2 PP} \\
\text{X2} &\rightarrow \text{Verb NP} \\
\text{VP} &\rightarrow \text{Verb PP} \\
\text{VP} &\rightarrow \text{VP PP} \\
\text{PP} &\rightarrow \text{Prep NP}
\end{aligned}
$$

$$
\begin{aligned}
\text{Det} &\rightarrow the \mid a \mid that \mid this \\
\text{Noun} &\rightarrow book \mid flight \mid meal \mid money \\
\text{Verb} &\rightarrow book \mid include \mid prefer \\
\text{Pronoun} &\rightarrow I \mid she \mid he \mid me \\
\text{PropNoun} &\rightarrow Singapore \mid Frankfurt \mid SIA \\
\text{Aux} &\rightarrow do \mid does \mid did \\
\text{Prep} &\rightarrow from \mid to \mid on \mid near \mid through \\
\text{S} &\rightarrow book \mid include \mid prefer \\
\text{VP} &\rightarrow book \mid include \mid prefer \\
\text{NP} &\rightarrow I \mid she \mid he \mid me \\
\text{NP} &\rightarrow Singapore \mid Frankfurt \mid SIA \\
\text{Nominal} &\rightarrow book \mid flight \mid meal \mid money
\end{aligned}
$$

Allowed format for all rules

$A \rightarrow \alpha$ with $A \in N,\quad \alpha \in (N \cup \Sigma)$

α **is either <u>1 terminal</u> OR <u>2 non-terminals</u>**

CFG to CNF — Summary

- Transformation of a CFG to a CNF
  - Every CFG can be transformed into a **weakly equivalent** CNF

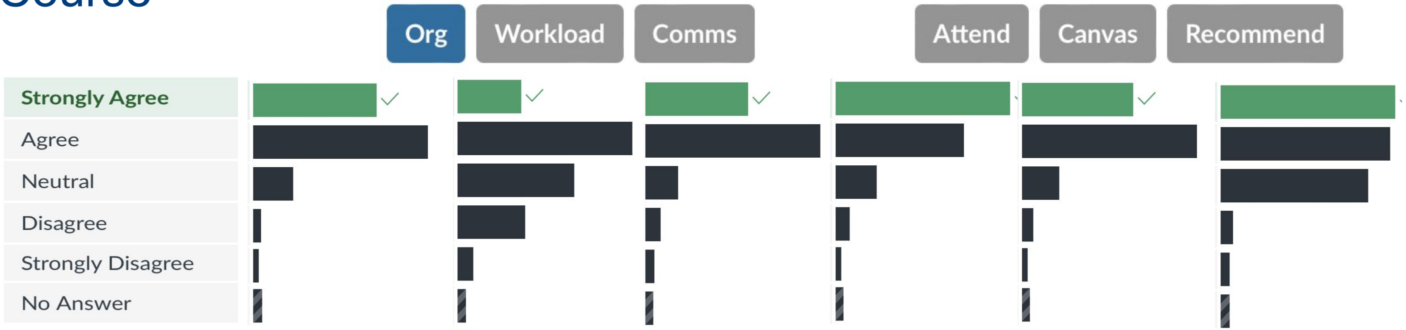
➜ **Weak equivalence**

- Two grammars generate the same set of sentences (identical expressiveness)
- The derivations generating the same sentences may differ
  (recall that the CNF may introduce additional non-terminals)

(Strong equivalence: identical expressiveness + identical derivations)

Midterm Feedback

Course

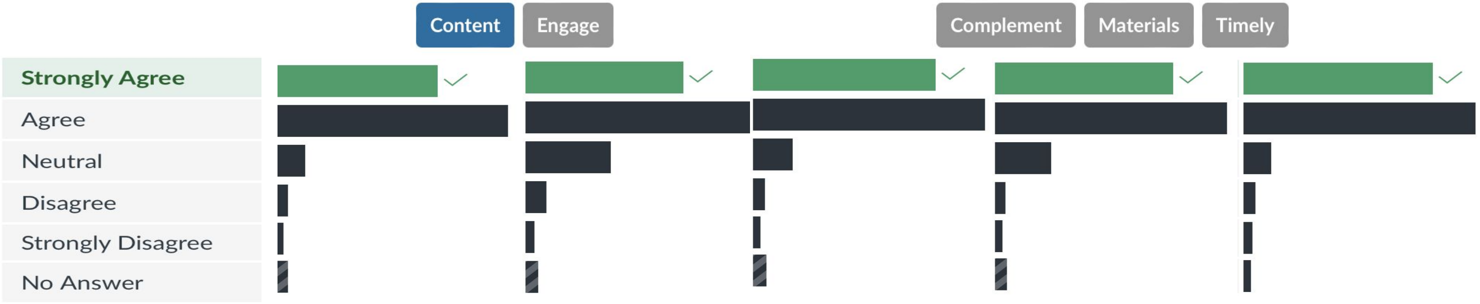


Lecturer

Midterm Feedback: Content

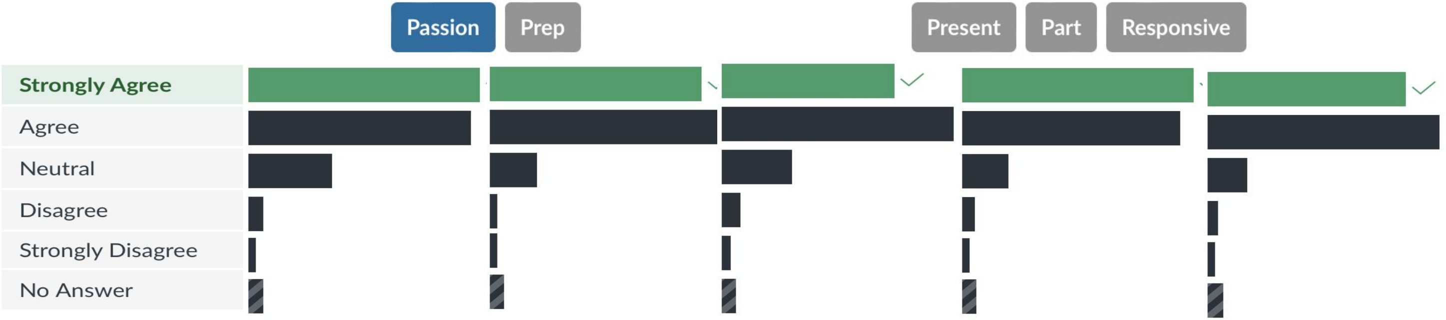
Lecture

Tutorial

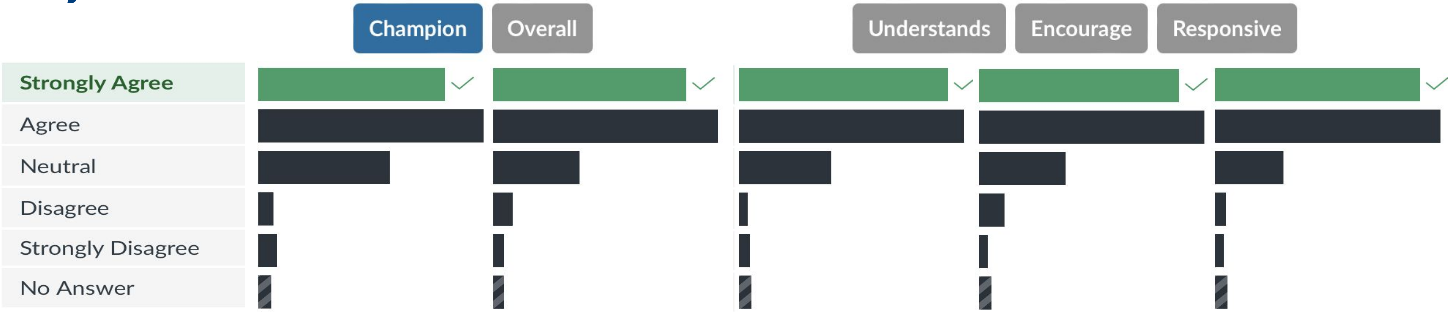


Midterm Feedback: Content

Tutorial Leader



Project Mentor



Improvements

Workload / Assignments

The course could benefit from slightly fewer assignments workload, considering that students also have other subjects to attend to. It's important to focus on the specificity and relevance of the tasks rather than their quantity and complexity, ensuring that each assignment meaningfully contributes to the learning experience.

The workload in this module does seem a bit much. The amount of effort required to do assignment 1 was a lot more than exepcted, and knowing that there were 2 more assignments with the same weightage was quite stressful.

Projects

I feel that the project component is severely lacking in guidance and structure. The only guidance we had was 1. datasets and 2. the rubrics. However, there's no reference point, or specific expectations for the standard we had to hit.

My friends and I found out that if we used the research question in the dataset.pdf, we might get a low grade due to how trivial it is, and we're expected to come up with our own research question, which was not clearly communicated.

Furthermore, having 6, potentially inexperienced people in a group for the project is difficult to manage, and a significant amount of time is spent coordinating the group, rather than the content itself.

Recordings / Length

I refer to recorded lecture material to consolidate and review some concepts I am unable to catch during the actual class. However, the recordings take a long time to be uploaded on Panapto, a week or more. Would appreciate if recordings were uploaded more timely,

The tutorial is too short, instructor often does not go through the entire tutorial sheet. There is also usually no time left at the end of the tutorial slot to go through any additional questions.

The ~3 hrs lecture is too long. Many students, in my honest opinion, do not appreciate the content well enough to be asking relevant questions during lectures.¬†

Outline

- Syntactic Parsing
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- **CYK Parsing Algorithm**
  - **Base membership algorithm**
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

CYK Parsing Algorithm

- CYK Parsing Algorithm — basic intuition
  - Given is a context-free grammar G in CNF
  - Assume we have a sentence W comprising n words

There can be multiple rules for different i, but at least one rule for at least one i.

G can generate W $\Leftrightarrow$ There exists a rule $\mathrm{A} \rightarrow \mathrm{B}\ \mathrm{C}$ in G with

- B can generate $w_1 w_2 w_3 w_4 \ldots w_i$
- C can generate $w_{i+1} \ldots w_{n-1} w_n$

binary split

$$\overbrace{w_1 w_2 w_3 w_4 \ldots w_i}^{B}\underbrace{w_{i+1} \ldots w_{n-1} w_n}_{C}$$

Note: Appreciate how the "binarization" of rules helps here!

CYK Parsing Algorithm

$$\overbrace{w_1 w_2 w_3 w_4 \dots w_i}^{B}\underbrace{w_{i+1} \dots w_{n-1} w_n}_{C}$$

- **Recursive nature:** G can generate B $\Longleftrightarrow$ There exists a rule $\mathrm{X} \rightarrow \mathrm{Y}\ \mathrm{Z}$ so that X and Y can generate a binary split of $w_1 w_2 w_3 \dots w_i$

  $\vdots$

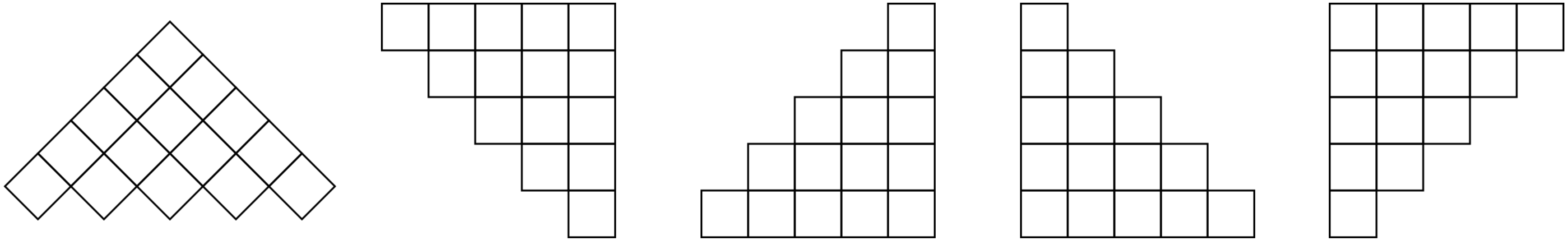
  (until we reach individual words; then check the lexicon rules)

- **CYK Parsing Algorithm: Solve problem using Dynamic Programming**
  - Find all possible parses for all sequences of size *k* for *k* from 1 to *n*

CYK Parsing Algorithm

- Dynamic Programming approach
  - Completing the parse table in a bottom-up manner
    (very similar idea as we have seen for calculating the Minimum Edit Distance)
  - Can to handle redundancy when computing the parse trees

- Different ways to visualize parse table
  - Completely identical idea, only the indexing of table cells will differ



CYK — Parse Table

[1,4] = all possible parses for span "book the flight"

I	book	the	flight	through	Singapore
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]
		[2,3]	[2,4]	[2,5]	[2,6]
			[3,4]	[3,5]	[3,6]
				[4,5]	[4,6]
					[5,6]

CYK parse table

- $N \times N$ table
  (N = #words in sentence)
- Each cell represents all the possible parses for span [i, j]
- Algorithm: fill table starting with cells for spans of length L = 1 to cells for spans of increasing lengths

L	Cells
1	[0,1], [1,2], [2,3], [3,4], [4,5], [5,6]
2	[0,2], [1,3], [2,4], [3,5], [4,6]
3	[0,3], [1,4], [2,5], [3,6]
4	[0,4], [1,5], [2,6]
5	[0,5], [1,6]
6	[0,6]

CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2]	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4]	[1,5]	[1,6]
		[2,3] Det	[2,4]	[2,5]	[2,6]
			[3,4] Nominal, Noun	[3,5]	[3,6]
				[4,5] Prep	[4,6]
					[5,6] PropNoun, NP

Cells for spans of length L = 1
➜ only need to check lexicon

Fill each cell with the non-terminals that can generate the corresponding word

$$
\begin{aligned}
Det &\rightarrow the \mid a \mid that \mid this \\
Noun &\rightarrow book \mid flight \mid meal \mid money \\
Verb &\rightarrow book \mid include \mid prefer \\
Pronoun &\rightarrow I \mid she \mid he \mid me \\
PropNoun &\rightarrow Singapore \mid Frankfurt \mid SIA \\
Aux &\rightarrow do \mid does \mid did \\
Prep &\rightarrow from \mid to \mid on \mid near \mid through \\
S &\rightarrow book \mid include \mid prefer \\
VP &\rightarrow book \mid include \mid prefer \\
NP &\rightarrow I \mid she \mid he \mid me \\
NP &\rightarrow Singapore \mid Frankfurt \mid SIA \\
Nominal &\rightarrow book \mid flight \mid meal \mid money
\end{aligned}
$$

CYK — Walkthrough

Cells for spans of length L > 1
➜ Check for each binary split if there is a production rule that can generate split

Example: Cell **[0,2]**
➜ only 1 binary split: [0,1] / [1,2]

Check each possible pair of non-terminals of binary split is the RHS of an existing production rule ➜ Yes, add LHS to cell

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4]	[1,5]	[1,6]
		[2,3] Det	[2,4] NP	[2,5]	[2,6]
			[3,4] Nominal, Noun	[3,5]	[3,6]
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

LHS	RHS
—	Pronoun S
—	Pronoun VP
—	Pronoun Nominal
—	Pronoun Noun
—	Pronoun Verb
—	NP S
S	**NP VP**
—	NP Nominal
—	NP Noun
—	NP Verb

Only this rule exists in our grammar

CYK — Walkthrough

Example: Cell **[1,4]**

- binary split: [1,2] / [2,4]
- binary split: [1,3] / [3,4]

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6]
		[2,3] Det	[2,4] NP	[2,5]	[2,6]
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Binary split: [1,2] / [2,4]

LHS	RHS
—	S NP
—	VP NP
—	Nominal NP
—	Noun NP
S, VP, X2	**Verb NP**

3 existing rules with the same RHS

Binary split: [1,3] / [3,4]

LHS	RHS

Empty because [1,3] is empty

CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6]
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Example: Cell **[2,6]**

- binary split: [2,3] / [3,6]
- binary split: [2,4] / [4,6]
- binary split: [2,5] / [5,6]

Binary split: [2,3] / [3,6]

LHS	RHS
NP	Det Nominal

Binary split: [2,4] / [4,6]

LHS	RHS
—	NP PP

Binary split: [2,5] / [5,6]

LHS	RHS

CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Example: Cell **[1,6]**

- binary split: [1,2] / [2,6]
- binary split: [1,3] / [3,6] **(empty!)**
- binary split: [1,4] / [4,6]
- binary split: [1,5] / [5,6] **(empty!)**

Binary split: [1,2] / [2,6]

LHS	RHS
—	S NP
—	VP NP
—	Nominal NP
—	Noun NP
S, VP, X2	**Verb NP**

Binary split: [1,4] / [4,6]

LHS	RHS
—	S PP
S, VP	**VP PP**
S, VP	**X2 PP**

CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Example: Cell **[0,6]**

- binary split: [0,1] / [1,6]
- binary split: [0,2] / [2,6]
- binary split: [0,3] / [3,6] **(empty!)**
- binary split: [0,4] / [4,6]
- binary split: [0,5] / [5,6] **(empty!)**

Binary split: [0,1] / [1,6]

LHS	RHS
—	Pronoun S
—	Pronoun VP
—	Pronoun X2
—	NP S
S	**NP VP**
—	NP X2

Binary split: [0,2] / [2,6]

LHS	RHS
—	S NP

Binary split: [0,4] / [4,6]

LHS	RHS
—	S PP

CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Our grammar can generate this sentence since the start symbol S is in [0,6]

CYK Parsing Algorithm — Pseudo Code

function CKY-Parse(*words*, *grammer*) **returns** *table*

for *j* ← **from** 1 **to** LENGTH(*words*) **do**
  for all { A | A→*words*[*j*] ∊ *grammar* }
    table[*j*-1, *j*] ← *table*[*j*-1, *j*] ∪ A

Base cases: For each terminal (i.e., word), find all terminals that can generator this terminal

for *j* ← **from** *j*-2 **down to** 0 **do**
  for *k*←*i*+1 to *j*-1 do
    for all { A | A→BC ∊ *grammar* **and** B ∊ *table*[*i*, *k*] **and** C ∊ *table*[*k*, *j*] }
      table[*i*, *j*] ← *table*[*i*, *j*] ∪ A

Loop over all possible binary splits of spans of size 2 and increasing

If there is a rule (or more) that can generate the current binary split, add LHS of rule to the cell of the current span

CYK Parsing Algorithm — Basic Python Implementation

```python
def cyk_parse(tokens, rules):
    n = len(tokens)

    # Initialize dynamic programming table
    CYK = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))

    # Initialize parse: span of length 1
    for s in range(n):
        # Find all non-terminals that can generate the terminal
        for A, rhs in rules:
            if rhs == (tokens[s],):
                CYK[s][s+1][A] = 1

    # Handle spans of length 2+ using dynamic programming
    for length in range(2, n+1):
        for start in range(0, n-length+1):     # Loop over all
            end = start + length                # the possible
            for split in range(start+1, end):  # binary splits
                # Check each production rule (ignore lexicon rules)
                for A, (B, C) in [ r for r in rules if len(r[1]) == 2]:
                    # is_valid = 1 if B and C can generate left and right part
                    is_valid = CYK[start][split][B] * CYK[split][end][C]
                    # The same LHS needs to be able to generate the RHS only once
                    CYK[start][end][A] = np.max([ is_valid, CYK[start][end][A] ])

    return CYK
```

CYK — Example: Invalid Parse

I	book	flight	the	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3] Nominal	[1,4]	[1,5]	[1,6]
		[2,3] Nominal, Noun	[2,4]	[2,5]	[2,6]
			[3,4] Det	[3,5]	[3,6]
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

CYK — Syntax vs. Semantic

- Syntactic parsing does not consider semantics
  - Any constituent can be replaced with another constituent of the same type
  - Example below: A noun can be replaced with any other noun

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

vs.

I	book	the	meal	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] VP, S, X2	[1,5]	[1,6] VP, S, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

CYK Parsing Algorithm — Limitation

- Basic CYK algorithm only solves the membership problem
  - Algorithm only checks if a sentence is a "member" of the language described by grammar

- What we also want
  - Finding all actual parse trees
    (in case a sentence is valid; otherwise the result is empty)
  - Identifying the best parse tree(s)
    (which requires a definition for what we mean by "best")

➜ Good news: Only rather minor extension to base algorithm required

In-Lecture Activity (10 mins incl. break)

Outline

- Syntactic Parsing
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- **CYK Parsing Algorithm**
  - Base membership algorithm
  - **Find all parse trees with backtracking**
  - Probabilistic parsing
  - Evaluation of parsers

CYK — Get all Parse Trees (Derivations)

- Basic Idea: Keep track of backtrace
  - Remember which 2 cells matched an existing production rule

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Binary split: [0,1] / [1,6]

LHS	RHS
—	Pronoun S
—	Pronoun VP
—	Pronoun X2
—	NP S
S	**NP VP**
—	NP X2

CYK — Get all Parse Trees (Derivations)

- Recall: Structural Ambiguity
  - In general, different production rules might match

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4] S	[0,5]	[0,6] S
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4] S, VP, X2	[1,5]	[1,6] S, VP, X2
		[2,3] Det	[2,4] NP	[2,5]	[2,6] NP
			[3,4] Nominal, Noun	[3,5]	[3,6] Nominal
				[4,5] Prep	[4,6] PP
					[5,6] PropNoun, NP

Binary split: [1,2] / [2,6]

LHS	RHS
—	S NP
—	VP NP
—	Nominal NP
—	Noun NP
S, VP, X2	Verb NP

Binary split: [1,4] / [4,6]

LHS	RHS
—	S PP
S, VP	VP PP
S, VP	X2 PP

CYK — Get all Parse Trees (Pseudo Code)

```
function CKY-Parse(words, grammer) returns table, pointer

    for j ← from 1 to LENGTH(words) do
        for all { A | A→words[j] ∈ grammar }
            table[j-1, j] ← table[j-1, j] ∪ A
            pointer[j-1, j, A] ← pointer[j-1, j, A] ∪ words[j]

    for j ← from j-2 down to 0 do
        for k←i+1 to j-1 do
            for all { A | A→BC ∈ grammar and B ∈ table[i, k] and C ∈ table[k, j] }
                table[i, j] ← table[i, j] ∪ A
                pointer[i, j, A] ← pointer[i, j, A] ∪ ((i, k, B), (k, j, C))
```

CYK — Get all Parse Trees (Python)

```python
def cyk_parse_basic_ptr(tokens, rules):
    n = len(tokens)

    # Initialize dynamic programming table + backtrace pointers
    CYK = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    PTR = defaultdict(lambda : defaultdict(lambda : defaultdict(list)))

    # Initialize parse: span of length 1
    for s in range(n):
        # Find all non-terminals that can generate the terminal
        for A, rhs in rules:
            if rhs == (tokens[s],):
                CYK[s][s+1][A] = 1
                PTR[s][s+1][A].append(tokens[s])

    # Handle spans of length 2+ using dynamic programming
    for length in range(2, n+1):
        for start in range(0, n-length+1):     # Loop over all
            end = start + length               # the possible
            for split in range(start+1, end):  # binary splits
                # Check each production rule (ignore lexicon rules)
                for A, (B, C) in [ r for r in rules if len(r[1]) == 2]:
                    # is_valid = 1 if B and C can generate left and right part
                    is_valid = CYK[start][split][B] * CYK[split][end][C]
                    # The same LHS needs to be able to generate the RHS only once
                    CYK[start][end][A] = np.max([ is_valid, CYK[start][end][A] ])
                    # If this is a valid split, remember from which cells we came
                    if is_valid > 0:
                        PTR[start][end][A].append(((start, split, B), (split, end, C)))

    return CYK, PTR
```

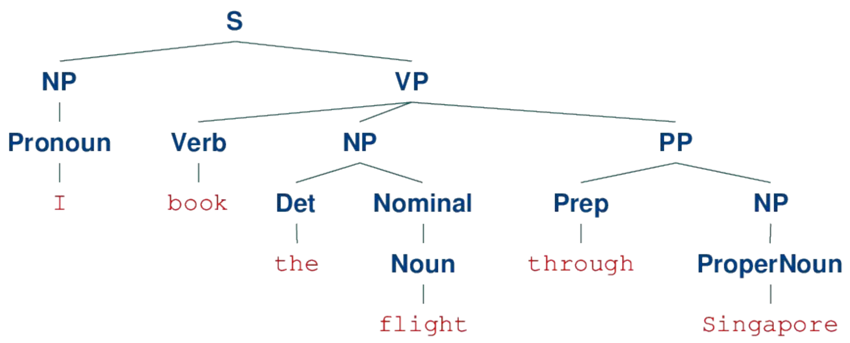
The only additions to the base algorithm

(base algorithm = CYK for membership problem)

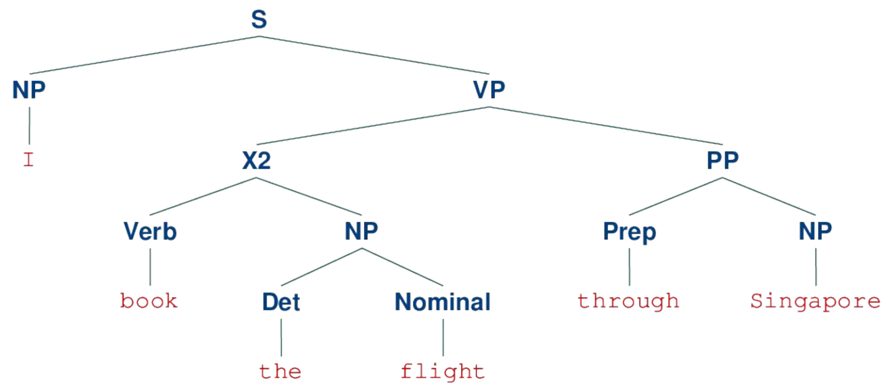
Parse Trees: CFG vs. CNF

- Converting a CFG into a CNF affects resulting parse trees
  - CFG parse trees can be recovered from CNF parse trees
    (easy for newly split n-ary rules; a bit more tricky for unary rules)
  - Straightforward extension of CYK algorithm to support unary rules directly
    (doesn't affect runtime complexity, but roughly doubles the required lines code)

Parse tree using original CFG



Parse tree using CNF (converted from CFG)

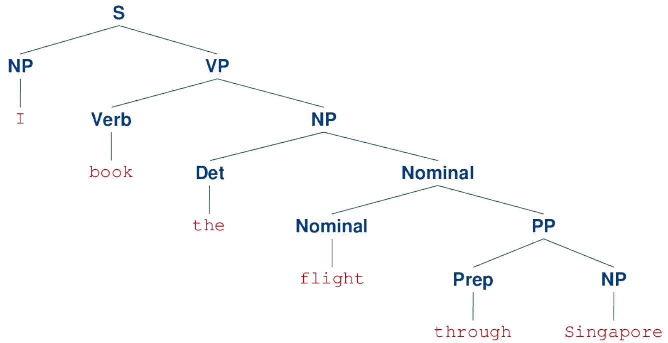


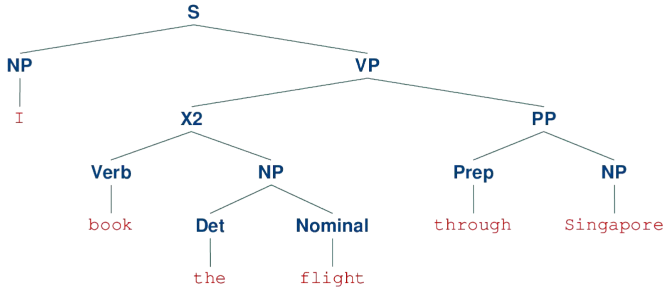
CYK — Parse Trees

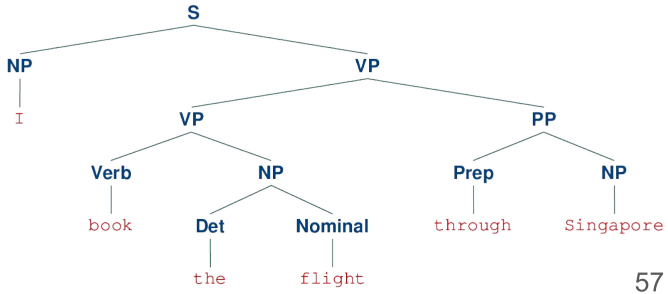
- Parse tree for example

  "I book the flight through Singapore"

- Observation
  - Multiple valid parses
  - Which is the best one?







Outline

- Syntactic Parsing
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- **CYK Parsing Algorithm**
  - Base membership algorithm
  - Find all parse trees with backtracking
  - **Probabilistic parsing**
  - Evaluation of parsers

Statistical Parsing

- Resolve structural ambiguity by choosing the most probable parse
  - Best parse = parse with the highest probability
  - Question: Where to get such probabilities from?

➜ Probabilistic Context-Free Grammar (PCFG)
  - Same as CFG, but each rule is associated with a probability
  - Probabilities are learned from an annotated dataset

Given a parse tree T for a sentence S comprised of rules: $$P(T, S) = \prod_{i}^{n} P(A \rightarrow \alpha) = \prod_{i}^{n} P(\alpha|A)$$

CFG — Formal Definition

- A CFG is a 4 tuple $\langle N, \Sigma, R, S \rangle$
  - N — set of non-terminal symbols
  - Σ — set of terminal symbols
  - R — set of rules

Allowed format for all rules

$$A \rightarrow \alpha \; [p] \quad \text{with} \quad A \in N, \;\; \alpha \in (N \cup \Sigma)$$
$$p = P(\alpha | A)$$

  - S — start symbol

Example

$$N = \{\mathrm{NP},\ \mathrm{VP},\ \mathrm{Det},\ \mathrm{Noun},\ \mathrm{Verb}\}$$

$$\Sigma = \{a,\ the,\ man,\ meal,\ flight,\ saw,\ booked\}$$

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{NP}\ \mathrm{VP}\ [0.4] \\ \mathrm{NP} &\rightarrow \mathrm{Det}\ \mathrm{Noun}\ [0.5] \\ \mathrm{VP} &\rightarrow \mathrm{Verb}\ \mathrm{NP}\ [0.2] \\ \mathrm{Det} &\rightarrow a \mid the\ [0.6] \\ &\ldots \end{aligned}$$

Example CFG → Example PCFG

$$\sum = 1 \left\{ \begin{array}{r} S \rightarrow \text{NP VP } [0.8] \\ S \rightarrow \text{Aux NP VP } [0.1] \\ S \rightarrow \text{VP } [0.1] \end{array} \right.$$

$$\sum = 1 \left\{ \begin{array}{r} \text{NP} \rightarrow \text{Pronoun } [0.2] \\ \text{NP} \rightarrow \text{ProperNoun } [0.2] \\ \text{NP} \rightarrow \text{Det Nominal } [0.6] \end{array} \right.$$

$$\sum = 1 \left\{ \begin{array}{r} \text{Nominal} \rightarrow \text{Noun } [0.3] \\ \text{Nominal} \rightarrow \text{Nominal Noun } [0.2] \\ \text{Nominal} \rightarrow \text{Nominal PP } [0.5] \end{array} \right.$$

$$\sum = 1 \left\{ \begin{array}{r} \text{VP} \rightarrow \text{Verb } [0.2] \\ \text{VP} \rightarrow \text{Verb NP } [0.4] \\ \text{VP} \rightarrow \text{Verb NP PP } [0.1] \\ \text{VP} \rightarrow \text{Verb PP } [0.1] \\ \text{VP} \rightarrow \text{VP PP } [0.2] \end{array} \right.$$

$$\text{PP} \rightarrow \text{Prep NP } [1.0]$$

$$\text{Det} \rightarrow \textit{the } [0.4] \mid \textit{a } [0.3] \mid \textit{that } [0.2] \mid \textit{this } [0.1]$$

$$\text{Noun} \rightarrow \textit{book } [0.2] \mid \textit{flight } [0.2] \mid \textit{meal } [0.3] \mid \textit{money } [0.3]$$

$$\text{Verb} \rightarrow \textit{book } [0.4] \mid \textit{include } [0.3] \mid \textit{prefer } [0.3]$$

$$\text{Pronoun} \rightarrow \textit{I } [0.4] \mid \textit{she } [0.2] \mid \textit{he } [0.2] \mid \textit{me } [0.2]$$

$$\text{ProperNoun} \rightarrow \textit{Singapore } [0.4] \mid \textit{Frankfurt } [0.4] \mid \textit{SIA } [0.2]$$

$$\text{Aux} \rightarrow \textit{do } [0.5] \mid \textit{does } [0.2] \mid \textit{did } [0.3]$$

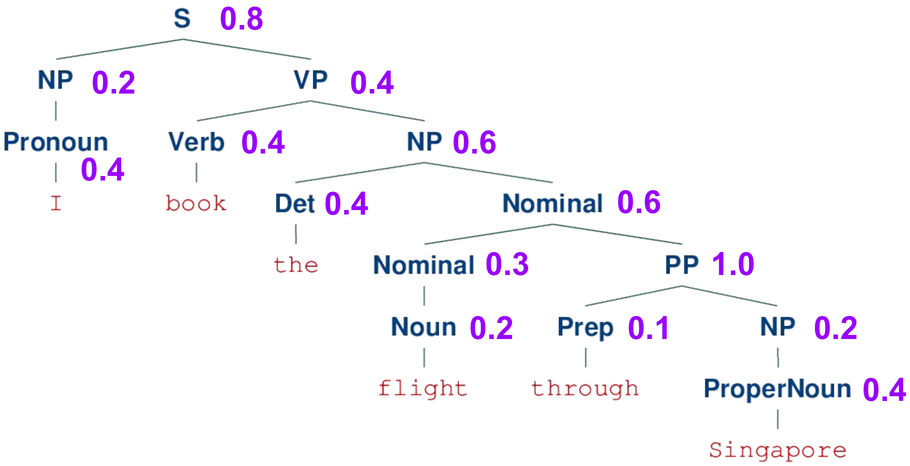
$$\text{Prep} \rightarrow \underbrace{\textit{from } [0.2] \mid \textit{to } [0.4] \mid \textit{on } [0.2] \mid \textit{near } [0.1] \mid \textit{through } [0.1]}_{\sum = 1 \text{ for all right-hand sides}}$$

Requirement for valid probabilities:

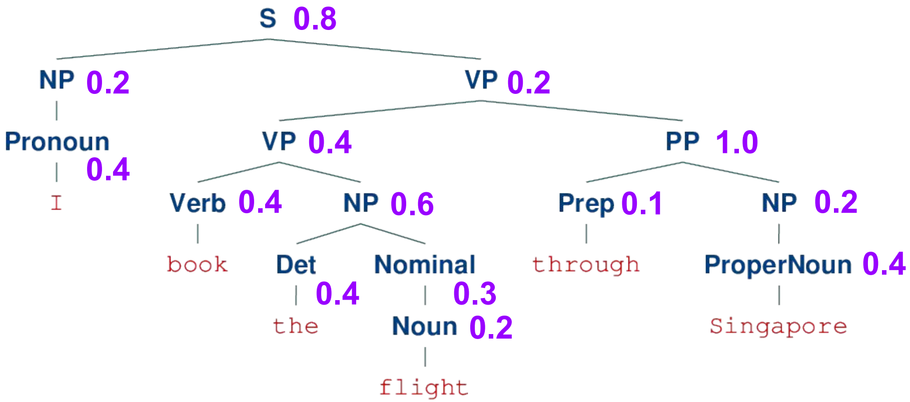
$$\sum_{\alpha} P(A \rightarrow \alpha) = \sum_{\alpha} P(\alpha | A) = 1$$

PCFG — Probability of a Parse Tree

- Probability of parse tree = product of probabilities of all rules
  - In practice, sum up log probabilities to avoid arithmetic underflow



$$P(T,S) = \prod_{i}^{n} P(A \rightarrow \alpha) = 0.00000071$$



$$P(T,S) = \prod_{i}^{n} P(A \rightarrow \alpha) = 0.00000024$$

PCFG — Calculating the Probability of a Rule

- Calculating $P(A \to \alpha)$ using Maximum Likelihood Estimation
  - Requires annotated dataset of parse trees

Number of occurrences of rule $P(A \to \alpha)$ in the dataset

$$P(A \to \alpha) = P(\alpha | A) = \frac{Count(A \to \alpha)}{Count(A)}$$

Number of occurrences of rules in the dataset with A as the LHS

PCFG — Converting to CNF

(1) Dividing n-ary rules by introducing new non-terminals

$$S \rightarrow \mathrm{NP}\ \mathrm{VP}\ [0.8]$$
$$S \rightarrow \mathrm{Aux}\ \mathrm{NP}\ \mathrm{VP}\ [0.1]$$
$$\ldots$$

$$S \rightarrow \mathrm{NP}\ \mathrm{VP}\ [0.8]$$
$$S \rightarrow \mathrm{X1}\ \mathrm{VP}\ [0.1]$$
$$\mathrm{X1} \rightarrow \mathrm{Aux}\ \mathrm{NP}\ [1.0]$$
$$\ldots$$

(2) Recursive removal of unary rules

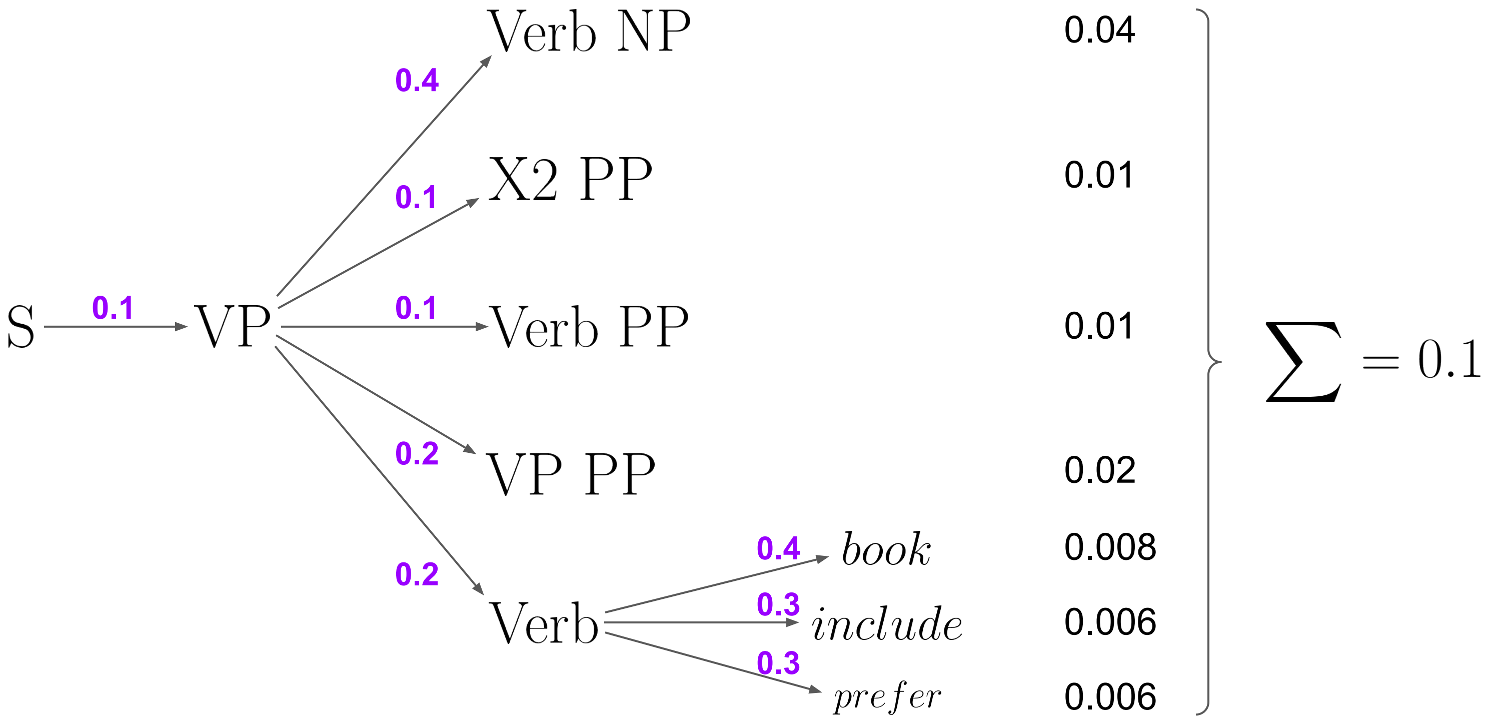
$$S \rightarrow \mathrm{NP}\ \mathrm{VP}\ [0.8]$$
$$S \rightarrow \mathrm{Aux}\ \mathrm{NP}\ \mathrm{VP}\ [0.1]$$
$$S \rightarrow \mathrm{VP}\ [0.1]$$
$$\ldots$$

$$S \rightarrow \mathrm{NP}\ \mathrm{VP}\ [0.8]$$
$$S \rightarrow \mathrm{X1}\ \mathrm{VP}\ [0.1]$$
$$\mathrm{X1} \rightarrow \mathrm{Aux}\ \mathrm{NP}\ [1.0]$$
$$S \rightarrow \mathrm{Verb}\ \mathrm{NP}\ [0.04]$$
$$S \rightarrow \mathrm{X2}\ \mathrm{PP}\ [0.01]$$
$$S \rightarrow \mathrm{Verb}\ \mathrm{PP}\ [0.01]$$
$$S \rightarrow \mathrm{VP}\ \mathrm{PP}\ [0.02]$$
$$S \rightarrow book\ [0.008] \mid include\ [0.006] \mid prefer\ [0.006]$$

How to compute these probabilities?

PCFG — Converting to CNF

- Multiply probabilities along the paths



PCFG — Converting to CNF

$$
\begin{aligned}
S &\rightarrow NP\ VP\ [0.8] \\
S &\rightarrow X1\ VP\ [0.1] \\
X1 &\rightarrow Aux\ NP\ [1.0] \\
S &\rightarrow Verb\ NP\ [0.04] \\
S &\rightarrow X2\ PP\ [0.01] \\
S &\rightarrow Verb\ PP\ [0.01] \\
S &\rightarrow VP\ PP\ [0.02] \\
NP &\rightarrow Det\ Nominal\ [0.6] \\
Nominal &\rightarrow Nominal\ Noun\ [0.2] \\
Nominal &\rightarrow Nominal\ PP\ [0.5] \\
VP &\rightarrow Verb\ NP\ [0.2] \\
VP &\rightarrow X2\ PP\ [0.1] \\
X2 &\rightarrow Verb\ NP\ [1.0] \\
VP &\rightarrow Verb\ PP\ [0.1] \\
VP &\rightarrow VP\ PP\ [0.2] \\
PP &\rightarrow Prep\ NP\ [1.0]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Det} &\rightarrow the\ [0.4] \mid a\ [0.3] \mid that\ [0.2] \mid this\ [0.1] \\
\mathrm{Noun} &\rightarrow book\ [0.2] \mid flight\ [0.2] \mid meal\ [0.3] \mid money\ [0.3] \\
\mathrm{Verb} &\rightarrow book\ [0.4] \mid include\ [0.3] \mid prefer\ [0.3] \\
\mathrm{Pronoun} &\rightarrow I\ [0.4] \mid she\ [0.2] \mid he\ [0.2] \mid me\ [0.2] \\
\mathrm{PropNoun} &\rightarrow Singapore\ [0.4] \mid Frankfurt\ [0.4] \mid SIA\ [0.2] \\
\mathrm{Aux} &\rightarrow do\ [0.5] \mid does\ [0.2] \mid did\ [0.3] \\
\mathrm{Prep} &\rightarrow from\ [0.2] \mid to\ [0.4] \mid on\ [0.2] \mid near\ [0.1] \mid through\ [0.1] \\
\mathrm{S} &\rightarrow book\ [0.008] \mid include\ [0.006] \mid prefer\ [0.006] \\
\mathrm{VP} &\rightarrow book\ [0.08] \mid include\ [0.06] \mid prefer\ [0.06] \\
\mathrm{NP} &\rightarrow I\ [0.08] \mid she\ [0.04] \mid he\ [0.04] \mid me\ [0.04] \\
\mathrm{NP} &\rightarrow Singapore\ [0.08] \mid Frankfurt\ [0.08] \mid SIA\ [0.04] \\
\mathrm{Nominal} &\rightarrow book\ [0.06] \mid flight\ [0.06] \mid meal\ [0.09] \mid money\ [0.09]
\end{aligned}
$$

CYK — Get Best Parse Tree (Pseudo Code)

```
function CKY-Parse(words, grammer, probs) returns table, pointer

    for j ← from 1 to LENGTH(words) do
        for all { A | A→words[j] ∈ grammar }
            table[j-1, j, A] ← probs[A→words[j]]
            pointer[j-1, j, A] ← pointer[j-1, j, A] ∪ words[j]

    for j ← from j-2 down to 0 do
        for k←i+1 to j-1 do
            for all { A | A→BC ∈ grammar and B ∈ table[i, k] and C ∈ table[k, j] }
                p ← table[i, k, B] * table[k, j, C] * probs[A→BC]
                if p > table[i, j, A] do
                    table[i, j, A] ← p
                    pointer[i, j, A] ← pointer[i, j, A] ∪ ((i, k, B), (k, j, C))
```

CYK — Get Best Parse Tree (Python)

```python
def cyk_parse_probabilistic_ptr(tokens, rules, probs):
    n = len(tokens)

    # Initialize dynamic programming table
    CYK = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    PTR = defaultdict(lambda : defaultdict(lambda : defaultdict(list)))

    # Initialize parse: span of length 1
    for s in range(n):
        # Find all non-terminals that can generate the terminal
        for A, rhs in rules:
            if rhs == (tokens[s],):
                CYK[s][s+1][A] = probs[A][token[s]]
                PTR[s][s+1][A].append(tokens[s])

    # Handle spans of length 2+ using dynamic programming
    for l in range(2, n+1):
        for start in range(0, n-l+1):
            end = start + l
            for split in range(start+1, end):
                # Check each production rule (ignore lexicon rules)
                for A, (B, C) in [ r for r in rules if len(r[1]) == 2]:
                    # Calculate probability of reaching the cell with the current rule
                    p = CYK[start][split][B] * CYK[split][end][C] * probs[A][(B,C)]
                    # If the probability is larger then the current one => update!
                    if p > CYK[start][end][A]:
                        CYK[start][end][A] = p
                        PTR[start][end][A].append(((start, split, B), (split, end, C)))

    return CYK, PTR
```

The only changes to the algorithm

Outline

- Syntactic Parsing
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- **CYK Parsing Algorithm**
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - **Evaluation of parsers**

Evaluation of Parse Trees

Note: This is not the only way to do it.

- Important: best parse $\nRightarrow$ correct parse
  - Best parse = parse with the highest probability
  - Correct parse = parse that matches the gold-standard solution

- How evaluate parse trees?
  - Represent each parse tree as a set of tuples $\{\langle l_1, i_1, j_1 \rangle, \langle l_2, i_2, j_2 \rangle, ..., \langle l_n, i_n, j_n \rangle\}$

    l_k is the non-terminal labeling the kth phrase

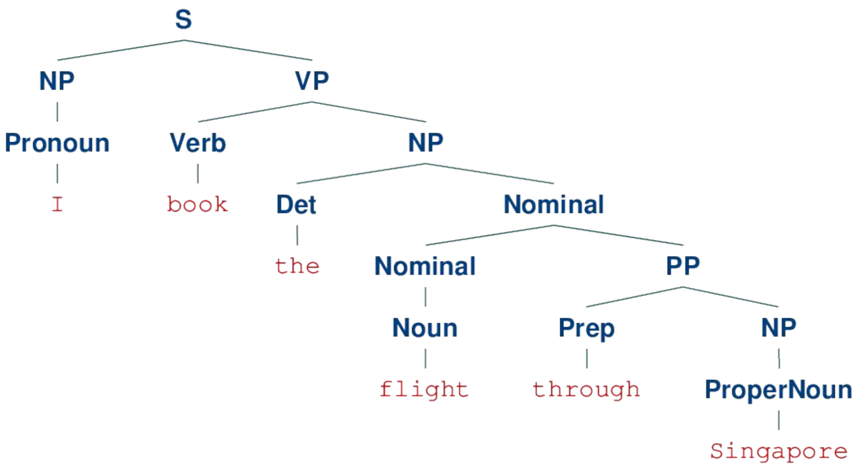
    i_k is the index of the first word in the kth word in the phrase

    j_k is the index of the last word in the kth word in the phrase

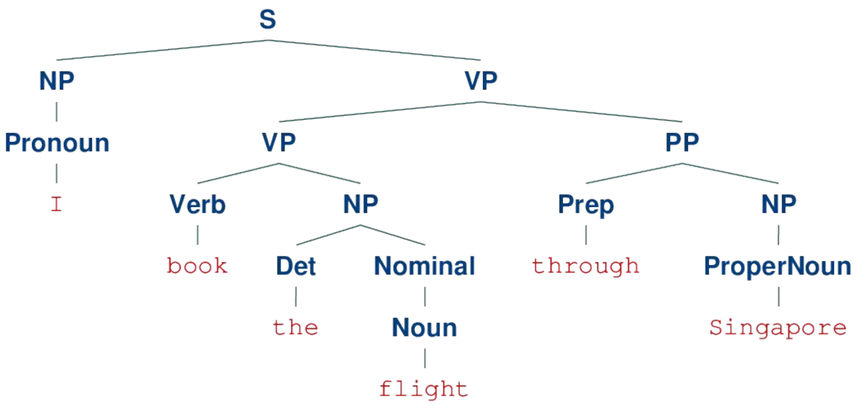
➜ Use representations of computed parse and gold standard parse to estimate precision, recall and f1

Evaluation of Parse Trees — Example

Gold-standard (correct) parse tree



Computed "best" parse tree



Tuples only present in correct tree

$\langle \mathrm{NP}, 3, 6 \rangle$

$\langle \mathrm{Nominal}, 4, 6 \rangle$

Tuples resent in both trees

$\langle \mathrm{NP}, 1, 1 \rangle$ $\langle \mathrm{Pronoun}, 1, 1 \rangle$ $\langle \mathrm{VP}, 2, 2 \rangle$ $\langle \mathrm{Verb}, 2, 2 \rangle$

$\langle \mathrm{Det}, 3, 3 \rangle$ $\langle \mathrm{Nominal}, 4, 4 \rangle$ $\langle \mathrm{Noun}, 4, 4 \rangle$ $\langle \mathrm{Prep}, 5, 5 \rangle$

$\langle \mathrm{ProperNoun}, 6, 6 \rangle$ $\langle \mathrm{PP}, 5, 6 \rangle$ $\langle \mathrm{NP}, 6, 6 \rangle$

Tuples only present in computed tree

$\langle \mathrm{VP}, 2, 4 \rangle$

$\langle \mathrm{NP}, 3, 4 \rangle$

Evaluation of Parse Trees — Example

Tuples only present in correct tree

$\langle \text{NP}, 3, 6 \rangle$

$\langle \text{Nominal}, 4, 6 \rangle$

Tuples resent in both trees

$\langle \text{NP}, 1, 1 \rangle$ $\langle \text{Pronoun}, 1, 1 \rangle$ $\langle \text{VP}, 2, 2 \rangle$ $\langle \text{Verb}, 2, 2 \rangle$

$\langle \text{Det}, 3, 3 \rangle$ $\langle \text{Nominal}, 4, 4 \rangle$ $\langle \text{Noun}, 4, 4 \rangle$ $\langle \text{Prep}, 5, 5 \rangle$

$\langle \text{ProperNoun}, 6, 6 \rangle$ $\langle \text{PP}, 5, 6 \rangle$ $\langle \text{NP}, 6, 6 \rangle$

Tuples only present in computed tree

$\langle \text{VP}, 2, 4 \rangle$

$\langle \text{NP}, 3, 4 \rangle$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{11}{11 + 2} = 0.85$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{11}{11 + 2} = 0.85$$

$$\text{f1} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = 0.85$$

TP = #tuples in both trees

FP = #tuples only in computed tree

FN = #tuples only in correct tree

Outline

- Syntactic Parsing
  - Quick recap: RegEx
  - Context-free grammars
  - Structural Ambiguity
  - Chomsky Normal Form

- CYK Parsing Algorithm
  - Base membership algorithm
  - Find all parse trees with backtracking
  - Probabilistic parsing
  - Evaluation of parsers

Summary

- Recursive nature of natural language
  - Natural language allows for nested structure
  - Basic building block: **constituents**
  - Most common way to capture constituency ➜ **context-free grammars (CFGs)**
- Syntactic parsing
  - Membership: check if a sentence can be generated by a grammar
  - Identification of <u>all</u> possible parse trees for a sentence
  - Identification of <u>best</u> parse tree for a sentence ➜ **Probabilistic CFGs**

CYK Parsing Algorithm

Pre-Lecture Activity for Next Week

Solutions to Quick Quizzes

Solutions to Quick Quizzes