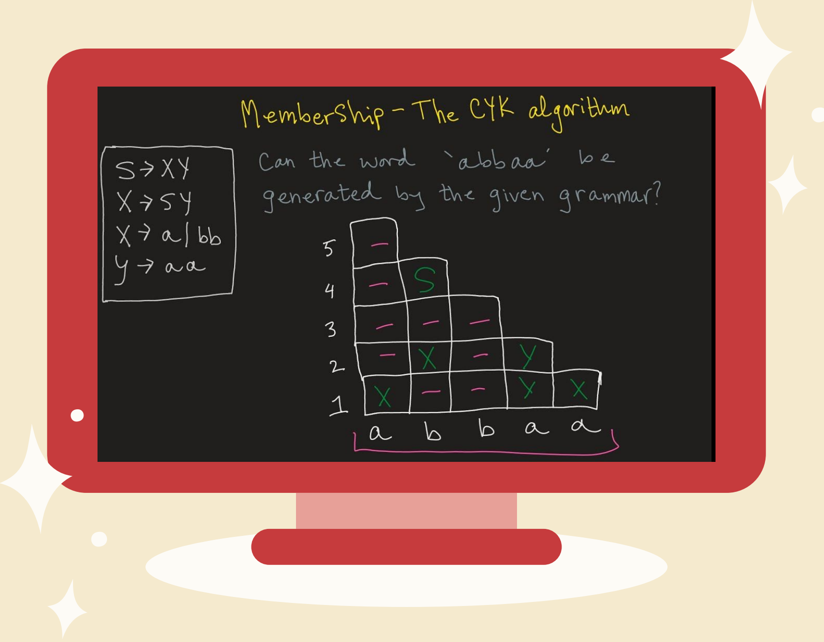
CYK Algorithm

Karis Moon and Daniel Park

At a Glance

The **CYK** (Cocke-Younger-Kasami) algorithm determines if a **string belongs to a language** defined by a <u>context-free grammar</u>



Requirement: CFG --> CNF

CNF (Chomsky Normal Form)

- A standard form for **all** Context-Free Grammars
- Simplifies parsing by ensuring compatibility by allowing to split current sequence into **two smaller sequences**

Requirements (for each production)

- Two non-terminals
- Single terminal
- Start goes to epsilon (empty string)

$$A \longrightarrow BC,$$
$$\text{or } A \longrightarrow a,$$
$$\text{or } S \longrightarrow \varepsilon,$$

CFG in CNF Steps

Step 1

Remove epsilon rules ($X \rightarrow \varepsilon$)

Step 2

Remove unit productions ($X \rightarrow Y$)

Step 3

Eliminate terminals in mixed rules ($X \rightarrow bC$)

Step 4

Break rules into binary rules

Conversion Example

S → ASB
A → aAS | a | ε
B → SbS | A | bb

-->

S0-> AS | PB | SB
S → AS | QB | SB
A → RS | XS | a
B → TS | VV | US | XS | a
X → a
Y → b
V → b
P → AS
Q → AS
R → XA
T → SY
U → XA

CYK Walkthrough

Given **string** *s* and **CFG** (in CNF form)

$S \rightarrow AB$

$S \rightarrow BC$

$A \rightarrow BA$

$A \rightarrow a$

$B \rightarrow CC$

$B \rightarrow b$

$C \rightarrow AB$

$C \rightarrow a$

5	S, A, C				
4	-	S, A, C			
3	-	B	B		
2	BA, BC / S, A	AA, AC, CA, CC / B	AB, CB / S, C	BA, BC / S, A	
1	B	A, C	A, C	B	A, C
s =	b	a	a	b	a

String *s* is in CFG

b, aaba
ba, aba
baa, ba
baab, a

B | S, A, C
S, A | B
A, C | B
BS, BA, BC,
SB, AB
SA, SC,
AB, CB,
BS, BA
BA, BC

Code implementation output

```
row 4

substring: baaba
split: ['b', 'aaba']
split first part origin: ['B']
split second part origin: ['S', 'A', 'C']
products: ['BS', 'BA', 'BC']
initial symbols: ['S', 'A']
split: ['ba', 'aba']
split first part origin: ['S', 'A']
split second part origin: ['B']
products: ['SB', 'AB']
initial symbols: ['S', 'A', 'C']
split: ['baa', 'ba']
split first part origin: []
split second part origin: ['S', 'A']
products: []
initial symbols: ['S', 'A', 'C']
split: ['baab', 'a']
split first part origin: []
split second part origin: ['A', 'C']
products: []
initial symbols: ['S', 'A', 'C']
processed dict updated: {'a': ['A', 'C'], 'b': ['B'], 'AB': ['S', 'C'], 'BC': ['S'], 'BA': [
], 'aba': ['B'], 'aaba': ['S', 'A', 'C'], 'baaba': ['S', 'A', 'C']}


FINAL ANSWER: baaba is in the given CFG, it can be obtained using these symbols: ['S', 'A', 'C']
```

CYK Complexity

Time Complexity: $O(n^3 \times |G|)$

Storing repeat substrings saves time

- *n:* length of input string
- *|G|:* size of grammar
- Filling in the table involves nested loots **iterating through all possible substrings of input string** to build up parsing decision

Space Complexity: $O(n^2)$

- Triangular table made to store the substrings of the string

Q&A

Thanks!

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**