



# CSCI 742 - Compiler Construction

---

Lecture 12

Cocke-Younger-Kasami (CYK) Algorithm

Instructor: Hossein Hojjat

February 20, 2017

## Recap: Chomsky Normal Form (CNF)

A CFG is in Chomsky Normal Form if each rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where

- $a$  is any terminal
- $A, B, C$  are non-terminals
- $B, C$  cannot be start variable

We allow the rule  $S \rightarrow \epsilon$  if  $\epsilon \in L$

## Recap: Conversion to CNF

1. remove unproductive non-terminals (optional)
2. remove unreachable non-terminals (optional)
3. make terminals occur alone on right-hand side
4. reduce arity of every production to  $\leq 2$
5. remove  $\epsilon$ -production rules  $X \rightarrow \epsilon$
6. remove unit productions  $X \rightarrow Y$
7. unproductive non-terminals
8. unreachable non-terminals

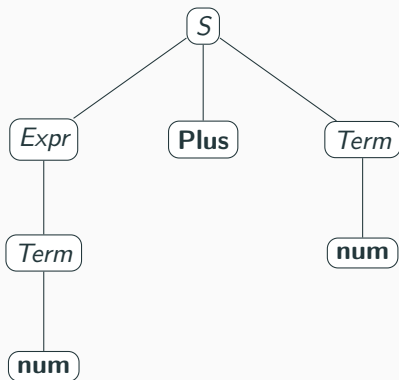
# Approaches to Parsing

## Top Down (Goal driven)

- Start from the start non-terminal  $S$
- Grow parse tree downwards to match the input word

## Bottom Up (Data Driven)

- Start from the input word
- Build up parse tree whose root is the start non-terminal  $S$



# CYK (Cocke-Younger-Kasami)

- One of the earliest recognition and parsing algorithms
- Bottom-up parsing (starts with words)
- Independently developed by J. Cocke, D. Younger, T. Kasami (CYK)
- Build solutions compositionally from sub-solutions
- Based on a “dynamic programming” approach

## CYK algorithm: Basic idea

For a grammar  $G$  and a word  $w$ :

- For every substring  $v_1$  of length 1, find all non-terminals  $A$  such that  $A \Rightarrow^* v_1$
- For every substring  $v_2$  of length 2, find all non-terminals  $A$  such that  $A \Rightarrow^* v_2$
- $\vdots$
- For the unique substring  $w$  of length  $|w|$ , find all non-terminals  $A$  such that  $A \Rightarrow^* w$

Check whether  $S$  belongs to the last set

- Standard CYK operates only on context-free grammars in Chomsky Normal Form (CNF)
- To illustrate the benefits of CNF we use a non-CNF grammar

# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 |  $\dots$  | 8 | 9

Sign  $\rightarrow$  + | -

1	2	.	3	e	+	4
---	---	---	---	---	---	---



# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

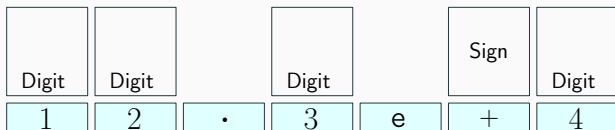
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 |  $\dots$  | 8 | 9

Sign  $\rightarrow$  + | -



- We start with substrings of length 1

# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

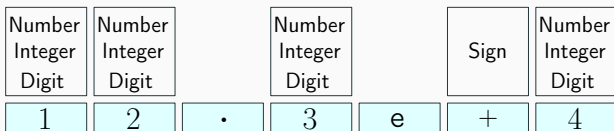
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 |  $\dots$  | 8 | 9

Sign  $\rightarrow$  + | -



- We start with substrings of length 1
- Grammar has **unit**-rules: after finding a non-terminal, we need to repetitively search for other non-terminals that derive the same word
- Chain of rules: Number  $\Rightarrow$  Integer  $\Rightarrow$  Digit

# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

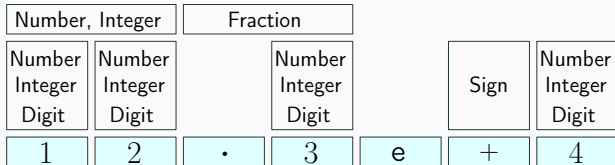
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 | ... | 8 | 9

Sign  $\rightarrow$  + | -



# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

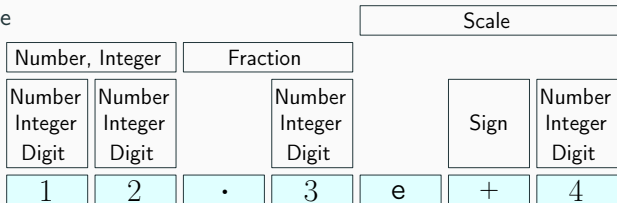
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 | ... | 8 | 9

Sign  $\rightarrow$  + | -



# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

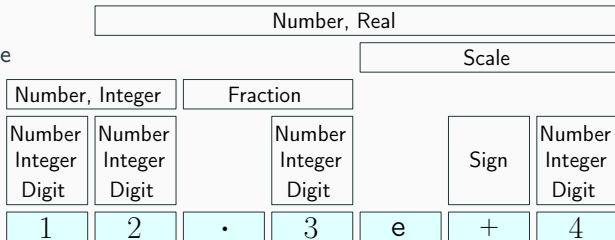
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 | ... | 8 | 9

Sign  $\rightarrow$  + | -



# Example word: 12.3e+4

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

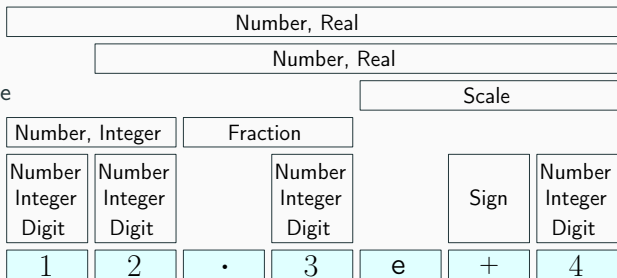
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 | ... | 8 | 9

Sign  $\rightarrow$  + | -



## Example word: 12.3

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$  e Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 |  $\dots$  | 8 | 9

Sign  $\rightarrow$  + | -

Number, Integer		Fraction	
Number Integer Digit	Number Integer Digit		Number Integer Digit
1	2	.	3

## Example word: 12.3

Number  $\rightarrow$  Integer | Real

Integer  $\rightarrow$  Digit | Integer Digit

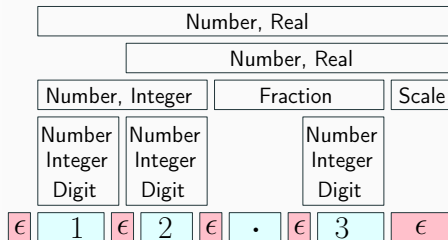
Real  $\rightarrow$  Integer Fraction Scale

Fraction  $\rightarrow$  . Integer

Scale  $\rightarrow$   $\epsilon$  Sign Integer |  $\epsilon$

Digit  $\rightarrow$  0 | 1 |  $\dots$  | 8 | 9

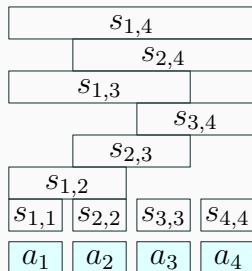
Sign  $\rightarrow$  + | -



- We neglected  $\epsilon$ -production rules until now
- $\epsilon$ -rules can be used between any two symbols, at the beginning and at the end



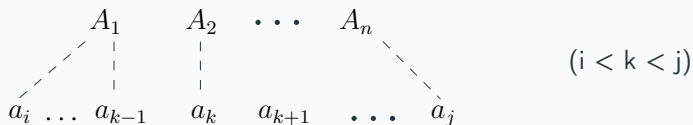
# Substring Recognition



- word:  $w = a_1a_2 \cdots a_n$
- $X_{i,j}$  set of all non-terminals deriving the substring  $s_{i,j}$
- Start non-terminal  $S$  derives  $w (= s_{1,n})$  if and only if  $S$  is a member of  $X_{1,n}$

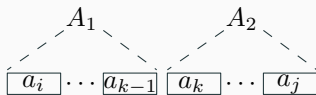
# Substring Recognition

- How to check if the RHS of rule  $A \rightarrow A_1 A_2 \dots A_n$  derives  $s_{i,j}$ ?

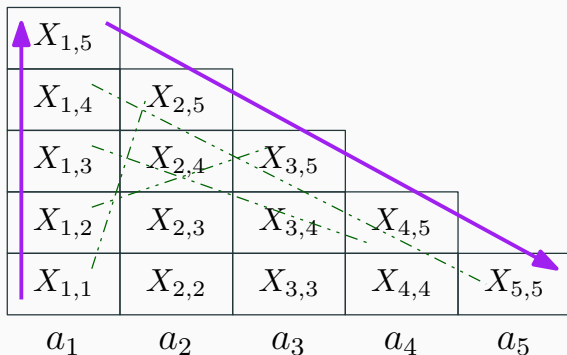


- We need to check all the possible matching of substrings with  $A_i$  non-terminals
- Results of checking subproblem solutions are reusable
- Subproblem results are computed once and then memoized for later

Matching in CNF:



# Recognition Table



Each row corresponds to one length of substrings

- Bottom Row: words of length 1
- Second from Bottom Row: words of length 2
- $\vdots$
- Top Row: word  $w$

# Example

**grammar:**

$$S \rightarrow AB \mid CB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow BC \mid b$$

$$C \rightarrow AC \mid a$$

**input:** *babab*

<i>B</i>	<i>A, C</i>	<i>B</i>	<i>A, C</i>	<i>B</i>
<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>

# Example

grammar:

$$S \rightarrow AB \mid CB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow BC \mid b$$

$$C \rightarrow AC \mid a$$

input: *babab*

<i>A, B</i>	<i>S</i>	<i>A, B</i>	<i>S</i>	
<i>B</i>	<i>A, C</i>	<i>B</i>	<i>A, C</i>	<i>B</i>
<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>

# Example

grammar:

$$S \rightarrow AB \mid CB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow BC \mid b$$

$$C \rightarrow AC \mid a$$

input: *babab*

<i>S</i>	<i>S</i>	<i>S</i>		
<i>A, B</i>	<i>S</i>	<i>A, B</i>	<i>S</i>	
<i>B</i>	<i>A, C</i>	<i>B</i>	<i>A, C</i>	<i>B</i>
<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>

# Example

grammar:

$$S \rightarrow AB \mid CB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow BC \mid b$$

$$C \rightarrow AC \mid a$$

input: *babab*

$S, A$					
$S$	$S$	$S$			
$A, B$	$S$	$A, B$	$S$		
$B$	$A, C$	$B$	$A, C$	$B$	
$b$	$a$	$b$	$a$	$b$	

# Example

grammar:

$$S \rightarrow AB \mid CB$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow BC \mid b$$

$$C \rightarrow AC \mid a$$

input: *babab*

<i>S</i>					
<i>S, A</i>					
<i>S</i>	<i>S</i>	<i>S</i>			
<i>A, B</i>	<i>S</i>	<i>A, B</i>	<i>S</i>		
<i>B</i>	<i>A, C</i>	<i>B</i>	<i>A, C</i>	<i>B</i>	
<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	



- What is the time and space complexity of CYK?
  - Assume size of input word is  $n$ , number of non-terminals is  $r$

- Parsers usually parse the original grammar without further modification
- Grammar reflects the actual structure of language
- Conversion to CNF can make it difficult to keep the intended structure