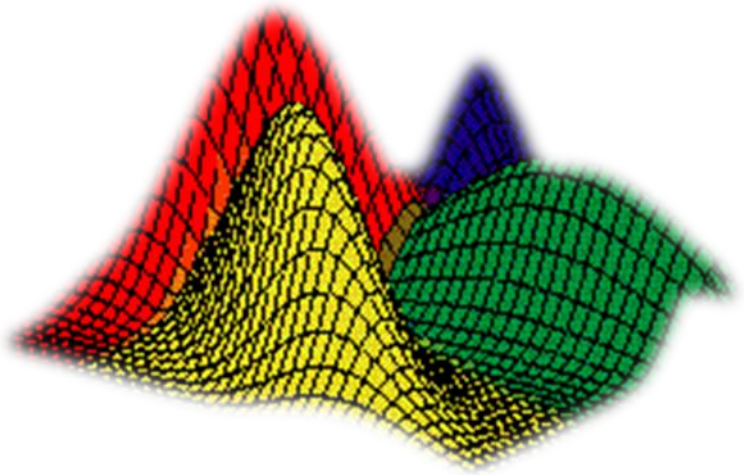


CSE 473: Pattern Recognition

Syntactic Pattern Recognition

- Objective:
  - Given a pattern description x,

    determine : $x \stackrel{?}{\in} L(G_i)$

SyntPR: A Simple Approach

- *Training*

  - For each class,
    - generate the grammar G_i
    - generate the Language $L(G_i)$
    - store the Language $L(G_i)$ in a database

- *Classification*
  - match the object pattern, x, with the patterns in each $L(G_i)$

Limitations in the Simple Approach

- $L(G_i)$ is often infinite
- Needs large storage
- Matching also computationally inefficient

Limitations in the Simple Approach

- $L(G_i)$ is often infinite
- Needs large storage
- Matching also computationally inefficient

Alternate solution:

Recognition by Parsing

Recognition by Parsing

- What we do in parsing
  - *Training*: Design a grammar for each class
  - *Classification*: Determine whether the object pattern, x, is *syntactically well formed* under a grammar

An Example of Parsing

- Grammar: $G = (V_T, V_N, P, S)$
- V_T = *(program, computer, crashes, the)*
- V_N = *(SENTENCE, ADJ, VP, NP, NOUN, VERB)*
- *P* = {

  SENTENCE -> NP + VP

  NP -> ADJ + NOUN

  VP -> VERB + NP

  NOUN -> computer | program

  VERB -> crashes

  ADJ -> the

  }

An Example of Parsing

- Determine:

  whether the sentence *"the program crashes the computer"* is valid under grammar *G*

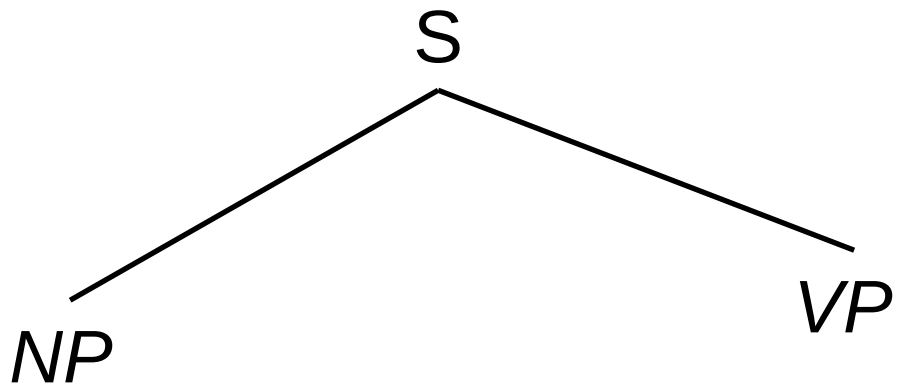
An Example of Parsing

- Determine:

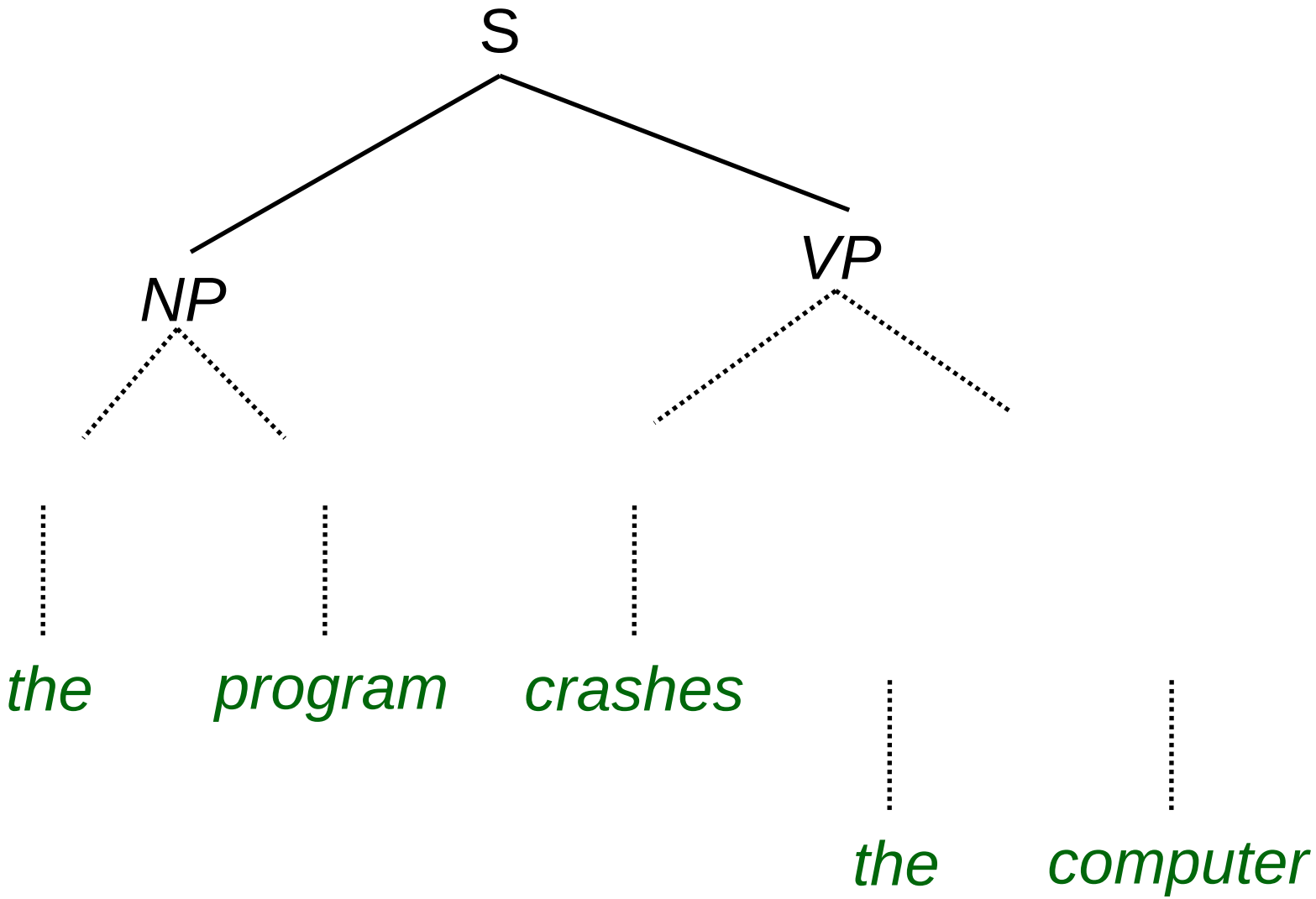
  whether the sentence *"the program crashes the computer"* is valid under grammar *G*

We will use a derivation tree

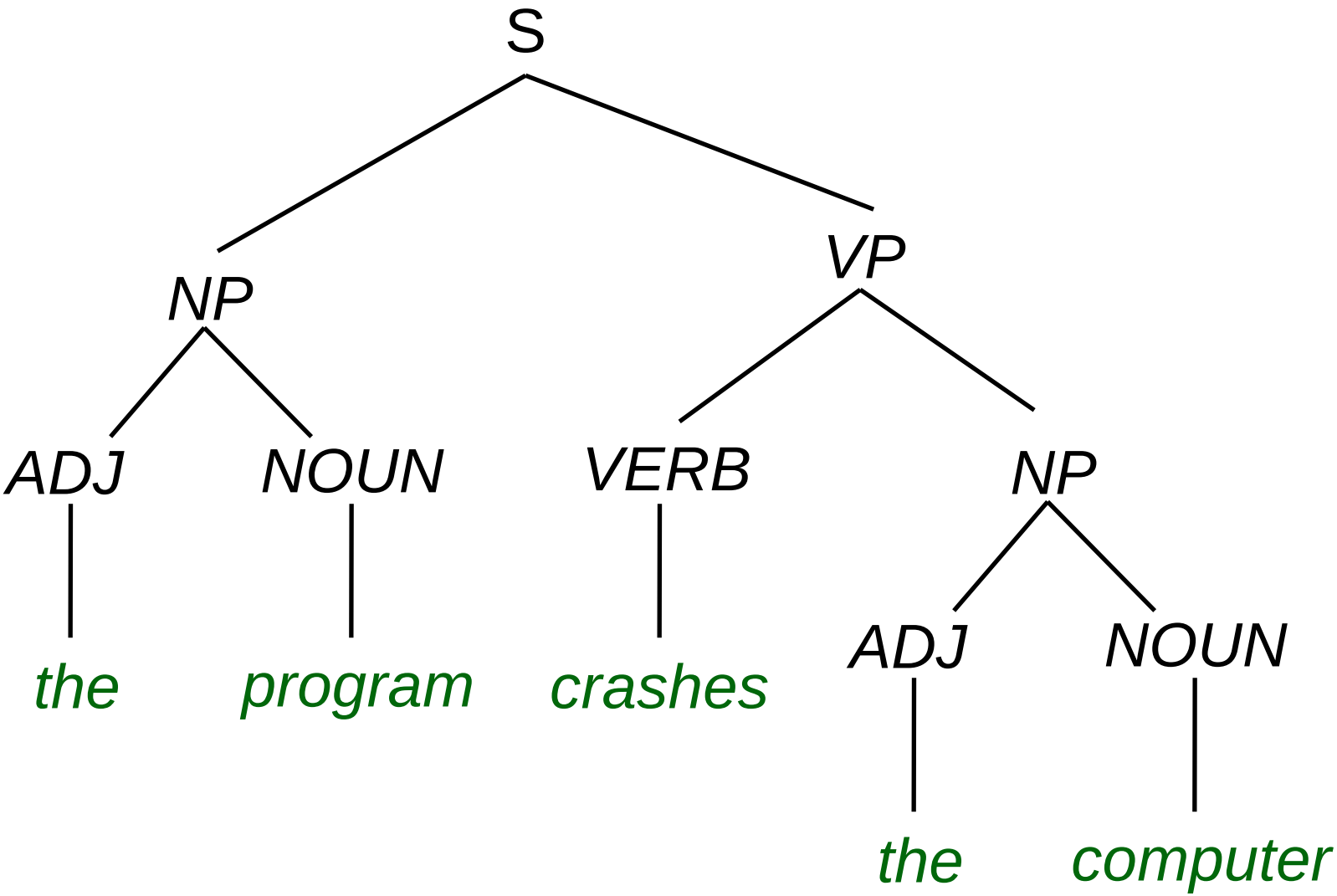
The Derivation Tree



The Derivation Tree

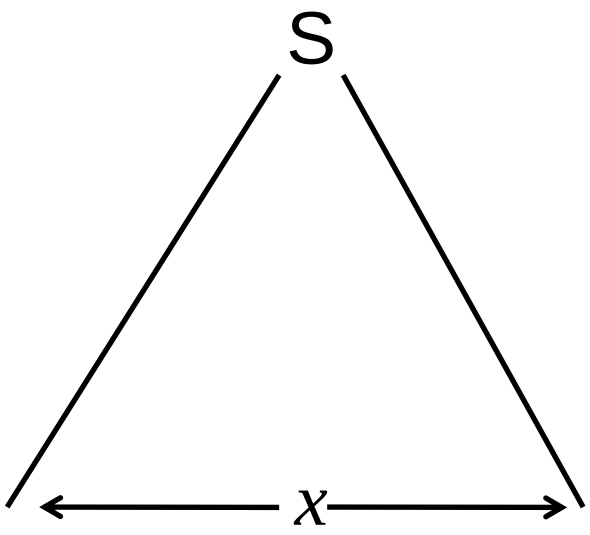


The Derivation Tree



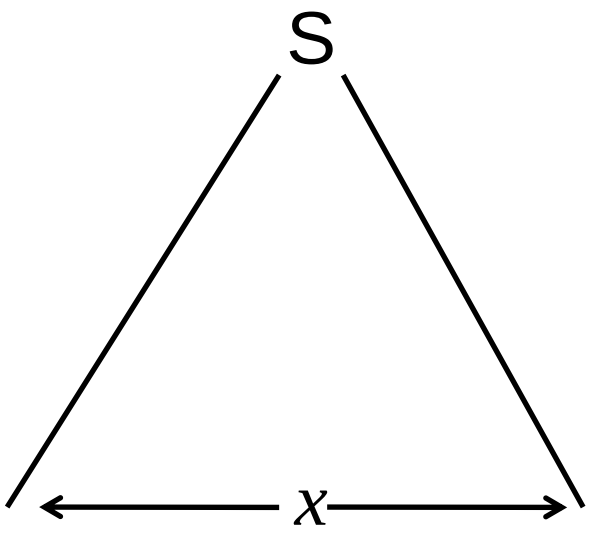
The Parsing Problem

- Given a grammar $G = (V_T, V_N, P, S)$ and a sentence x, form a triangle like this:



The Parsing Problem

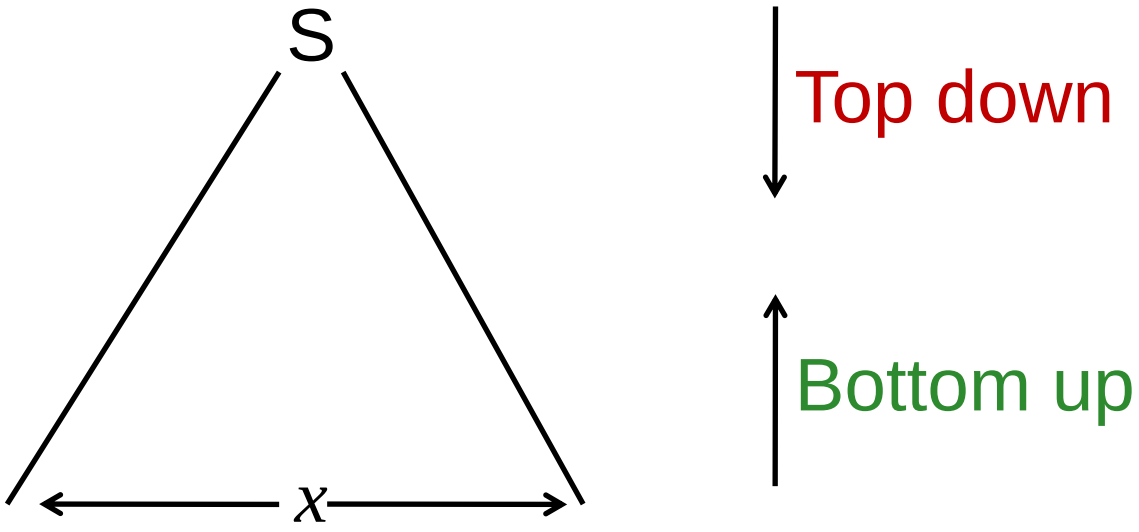
- Given a grammar $G = (V_T, V_N, P, S)$ and a sentence x, form a triangle like this:



- The parsing is the process of filling of interior of the triangle

The Parsing Problem

- Use either top down or bottom up approach



- Top down: proceed from *S* towards terminals
- Bottom up: proceed from terminals towards *S*

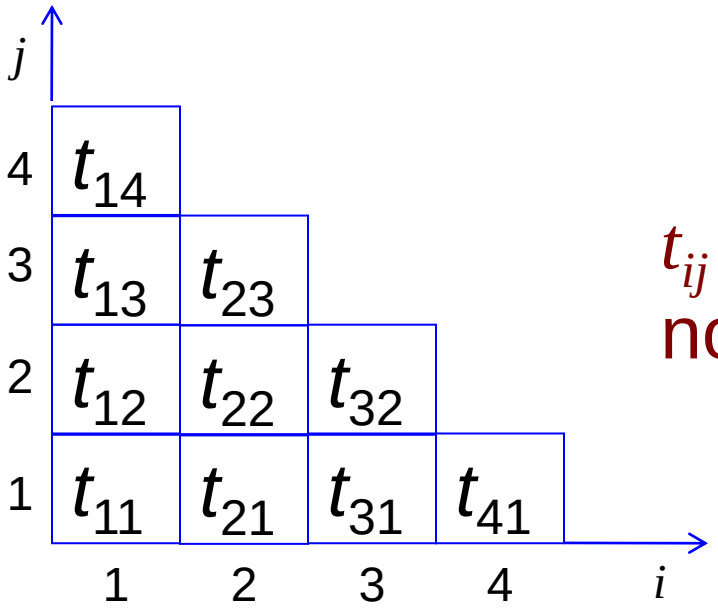
Cocke-Younger-Kasami (CYK) Parsing Algorithm

- A bottom up approach
- Complexity is $O(|x|^3)$

- Requirements:
  - The productions rules must be in CNF

CYK Parsing Algorithm

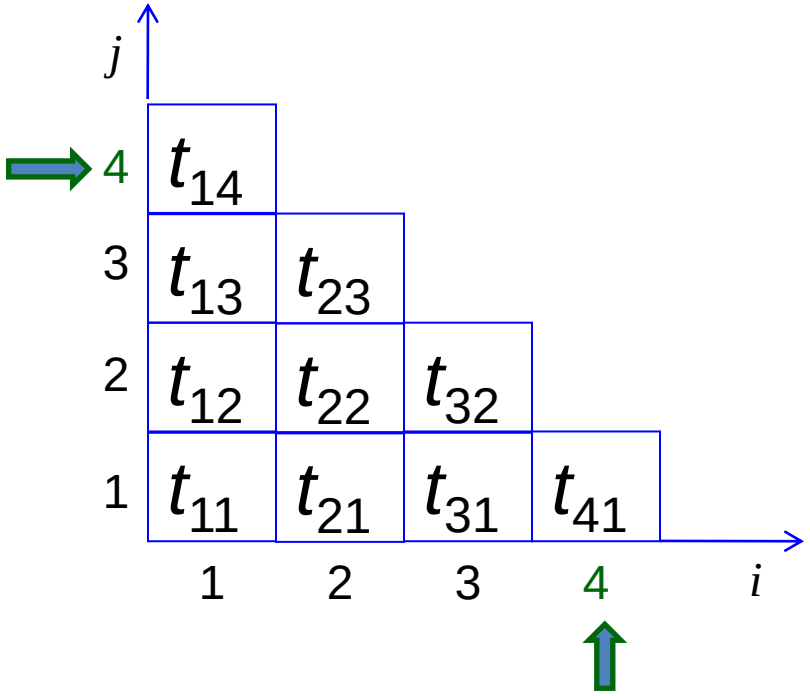
- Given a grammar G and a string $x = x_1, x_2, \ldots, x_n$, form a table like this:



t_{ij} is a set of non-terminals

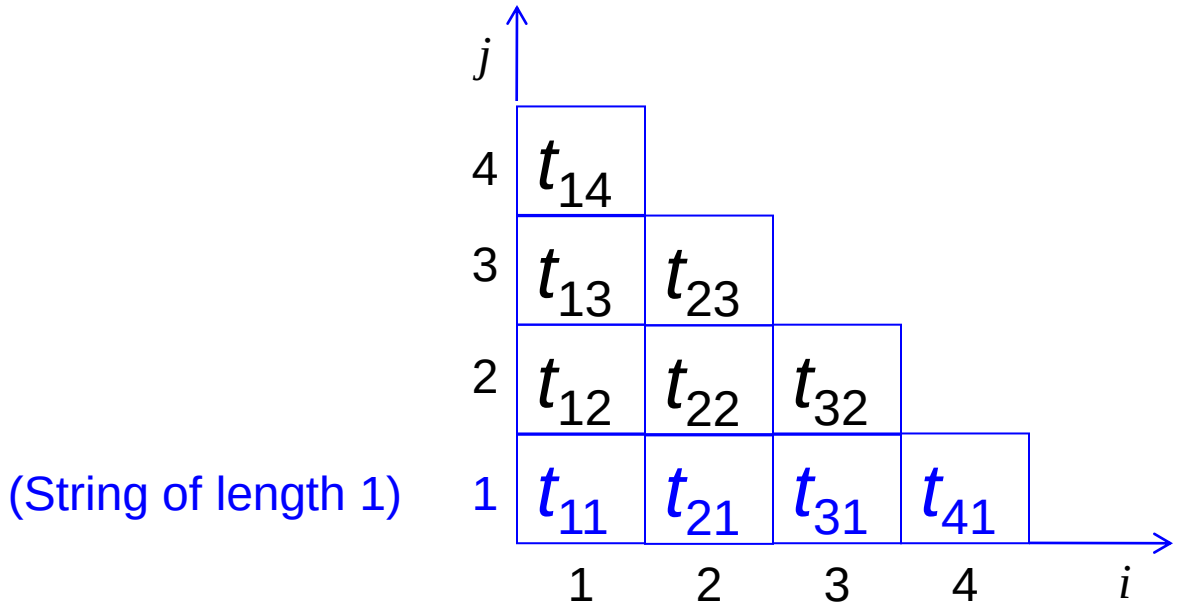
CYK Parsing Algorithm

- Given a grammar *G* and a string $x = x_1, x_2, \ldots, x_n$, form a table like this:
  - The number of rows/columns = n



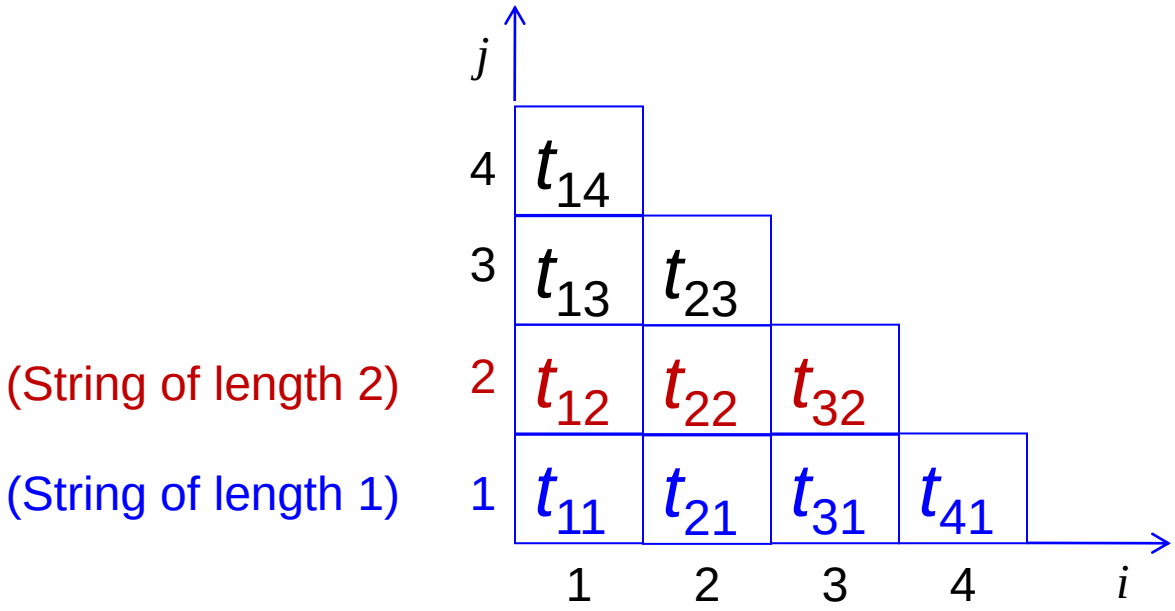
CYK Parsing Algorithm

- String covering



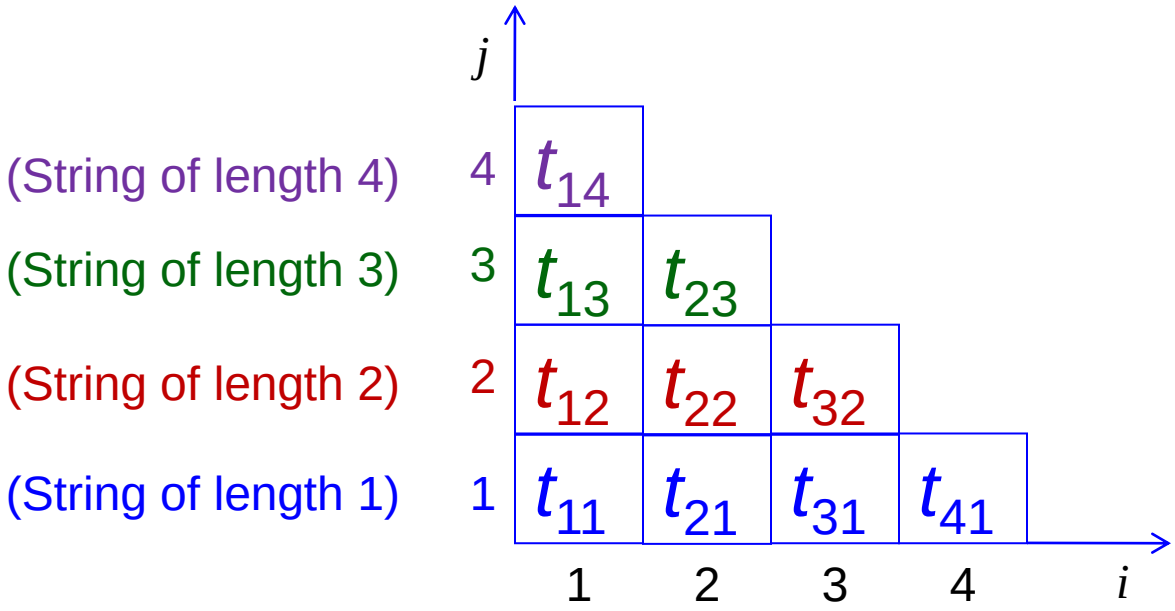
CYK Parsing Algorithm

- String covering



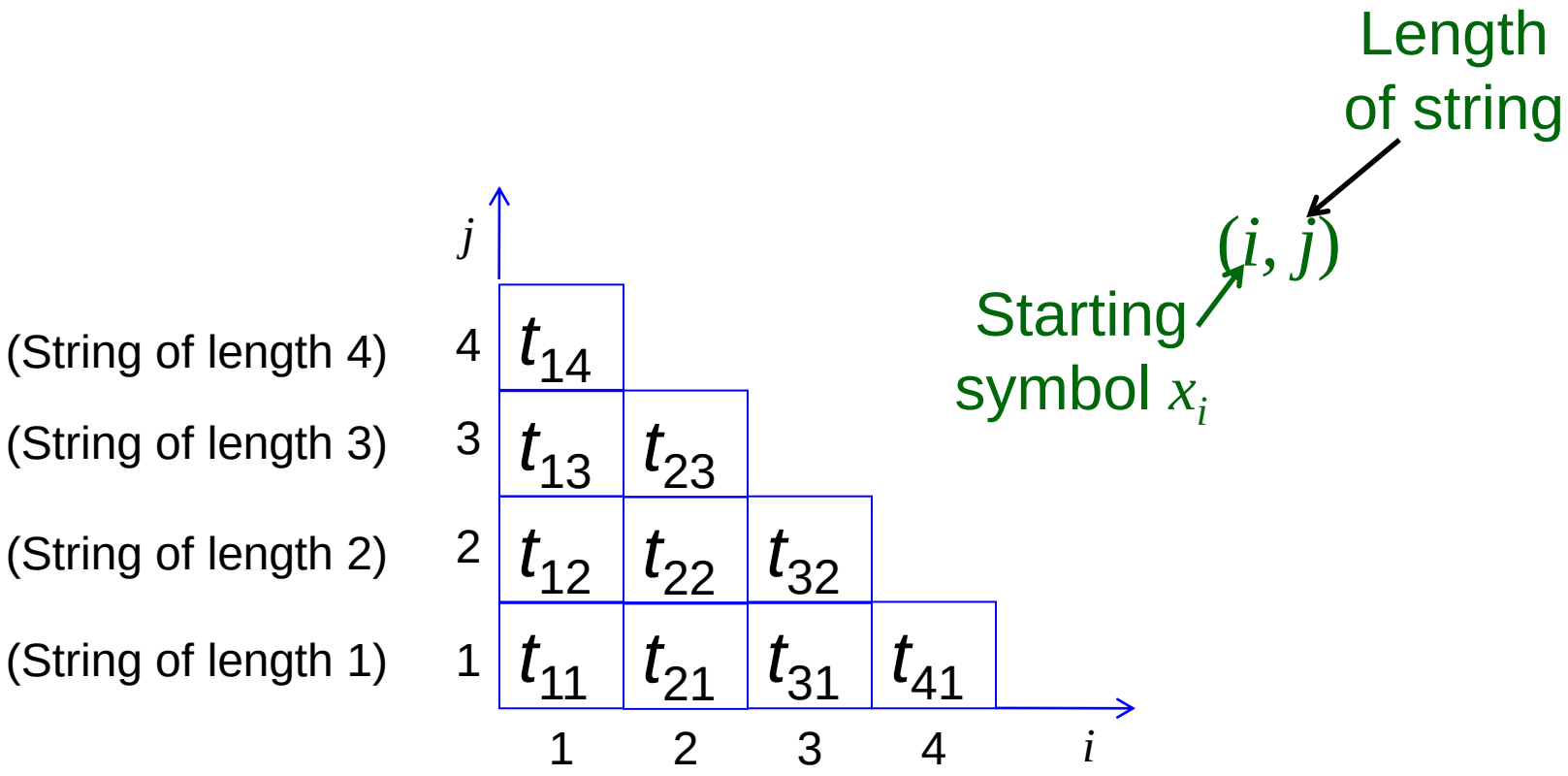
CYK Parsing Algorithm

- String covering



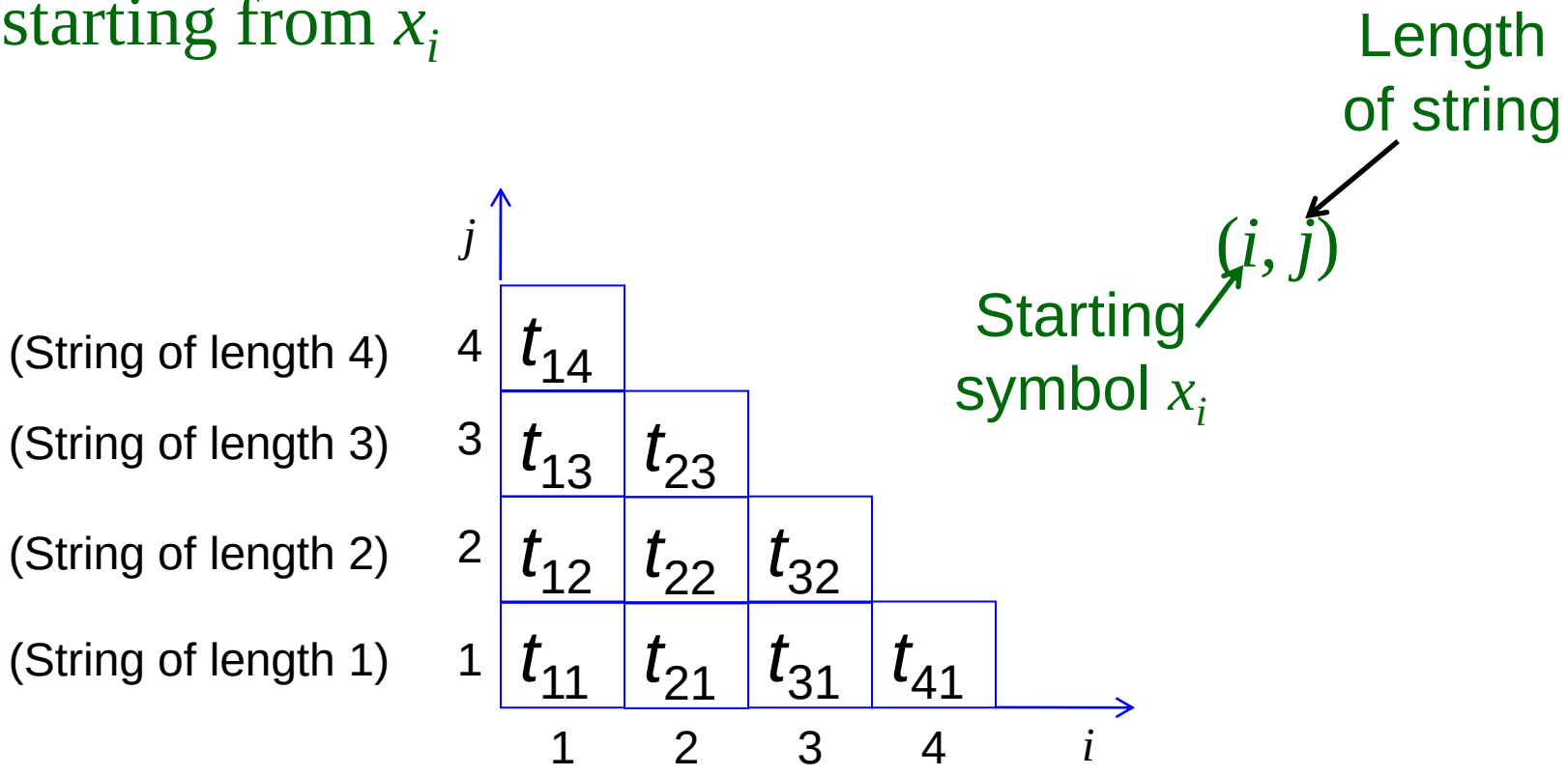
CYK Parsing Algorithm

- String covering by entry t_{ij}



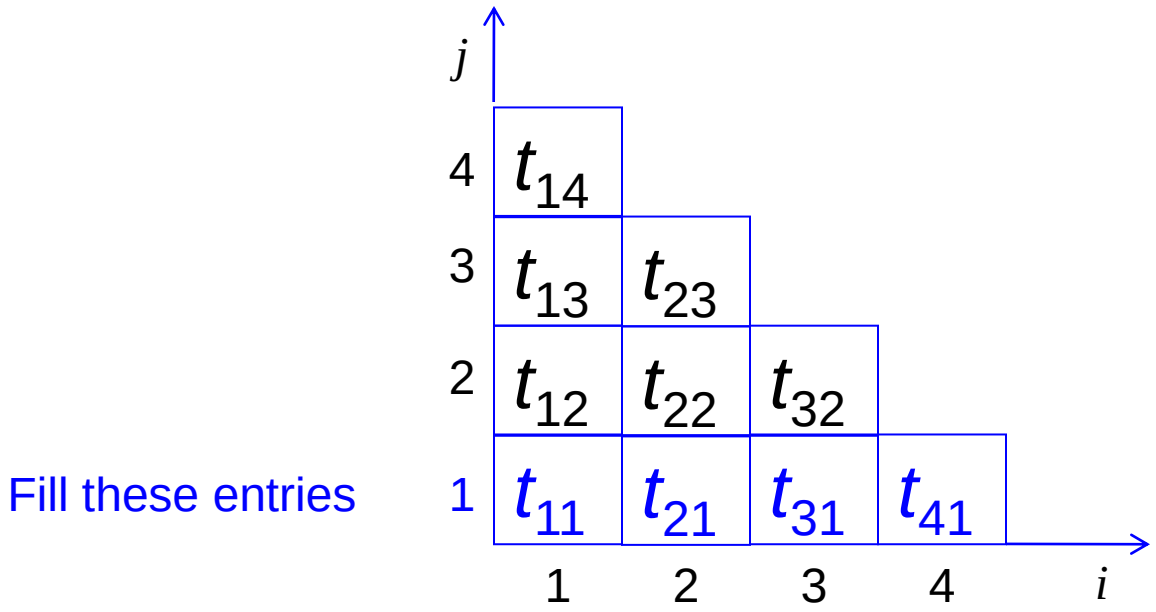
CYK Parsing Algorithm

- String covering by entry t_{ij}
  - A non-terminal in t_{ij} covers string of length j starting from x_i



CYK Parsing Algorithm

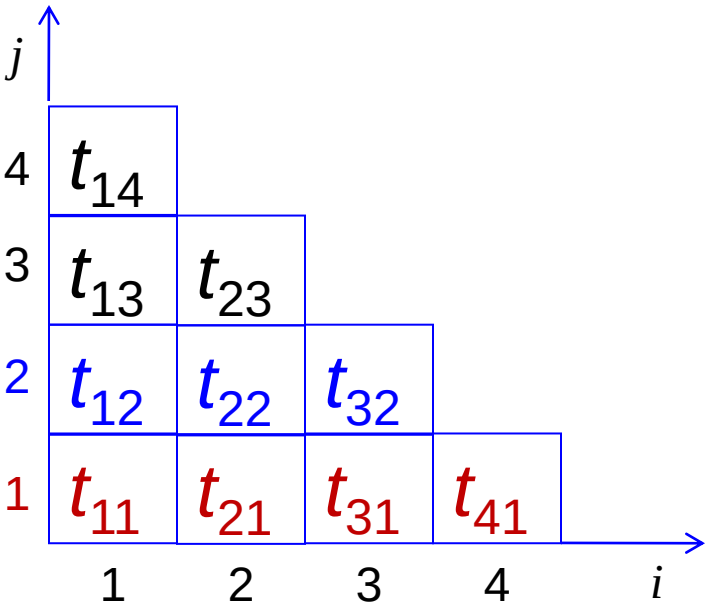
- Filling CYK table



CYK Parsing Algorithm

- Filling CYK table

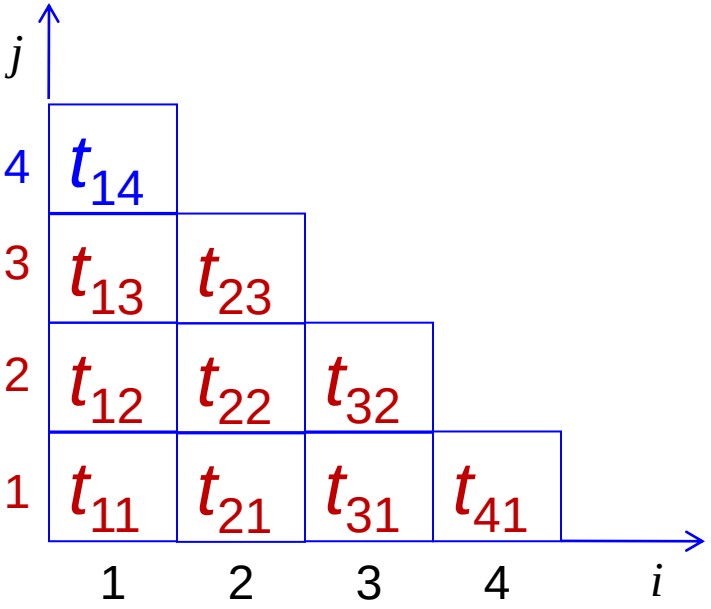
Fill these entries



CYK Parsing Algorithm

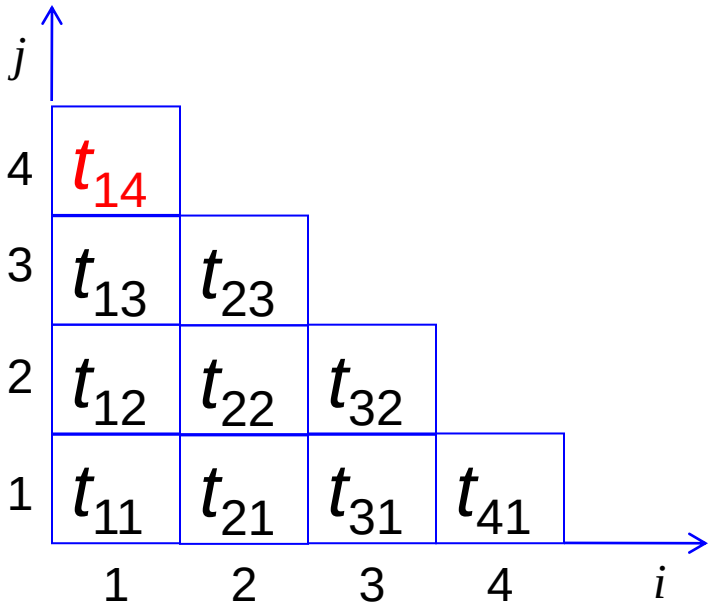
- Filling CYK table

Fill these entries



CYK Parsing Algorithm

- With a grammar *G* and a string $x = x_1, x_2, \ldots, x_n$
- If t_{1n} (t_{14}, in this example) contains *S*, this means, x is defined in G



CYK Parsing Algorithm

- Example:

  Let a grammar with following productions

  S -> AB | BB, *A -> CC | AB | a*

  B -> BB |CA | b, *C -> BA | AA | b*

CYK Parsing Algorithm

- Example:

  Let a grammar with following productions

  S -> AB | BB, *A -> CC | AB | a*

  B -> BB |CA | b, *C -> BA | AA | b*

now parse, x = *aabb*

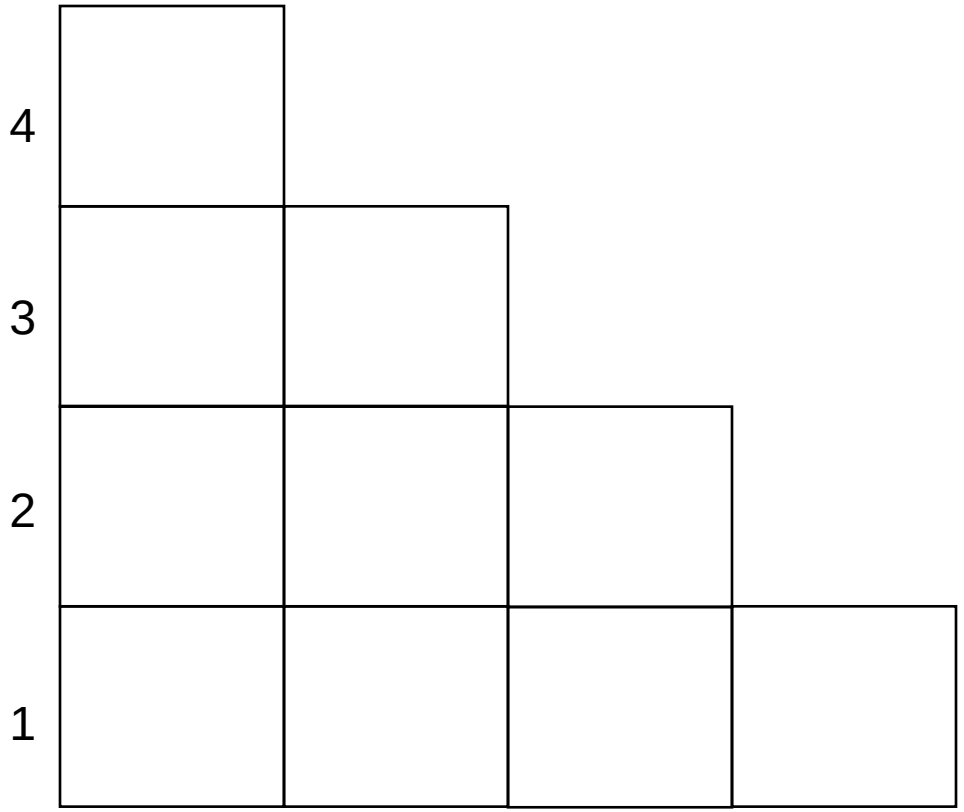
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



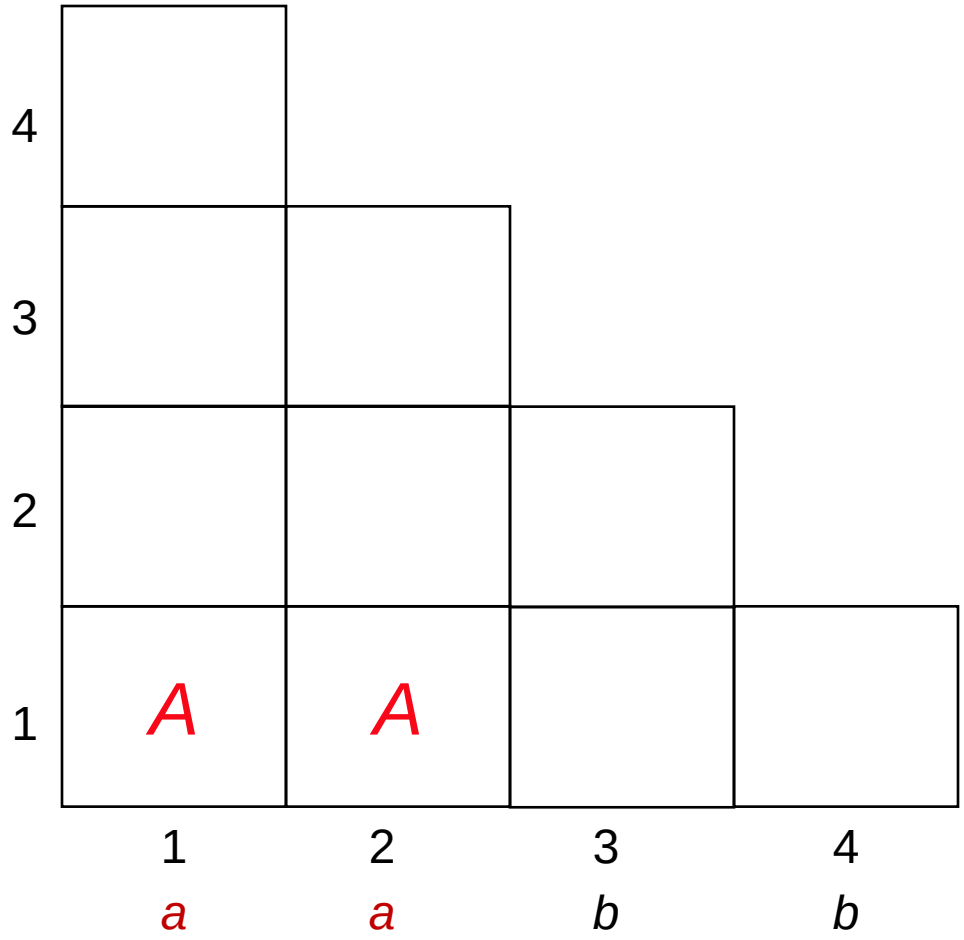
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



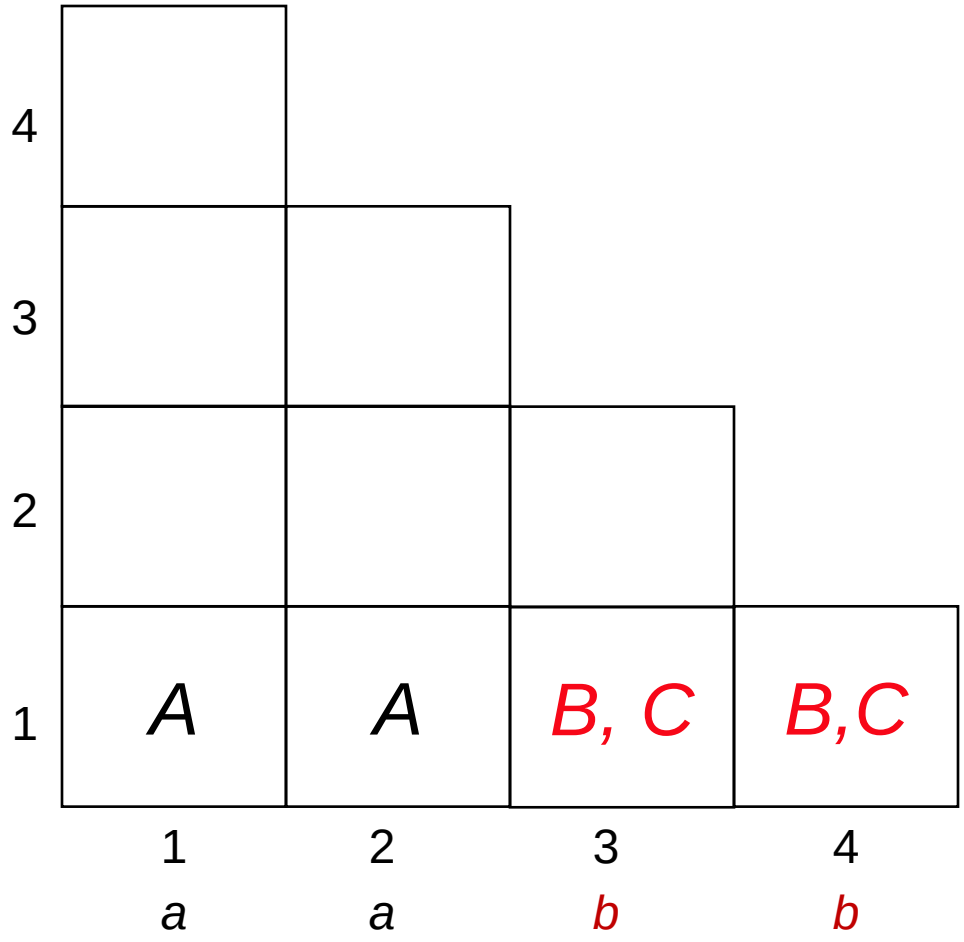
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



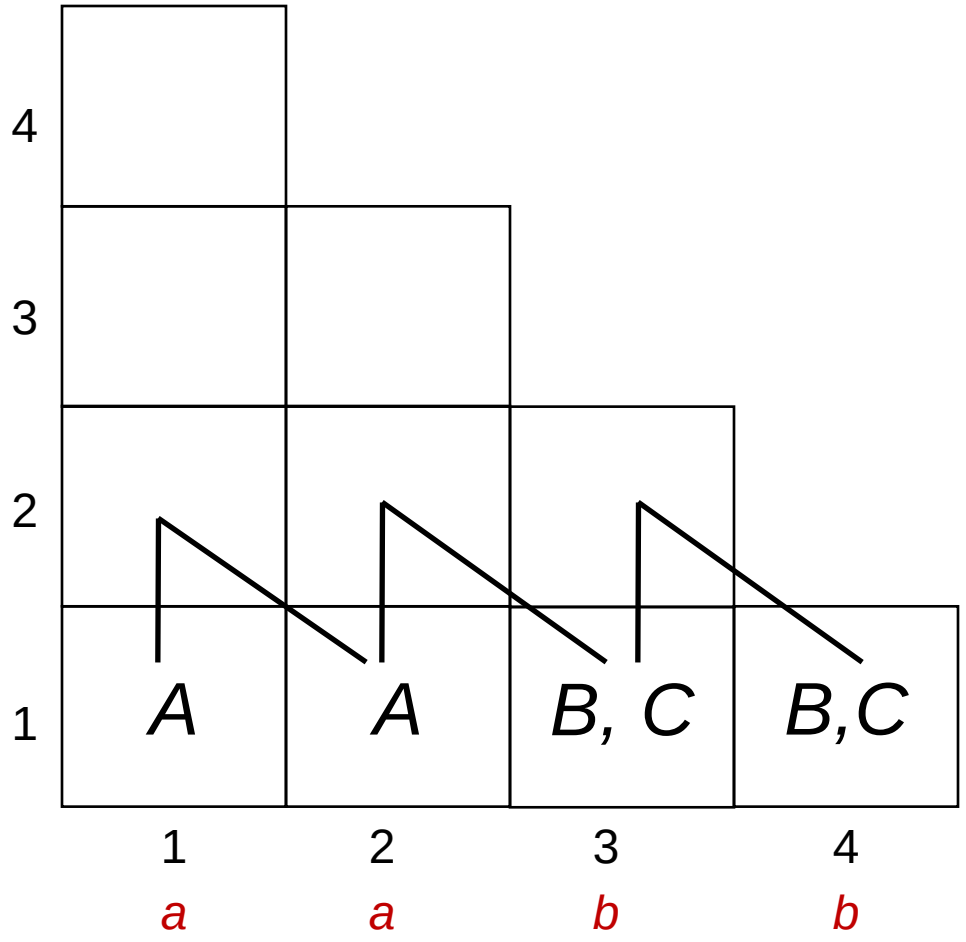
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



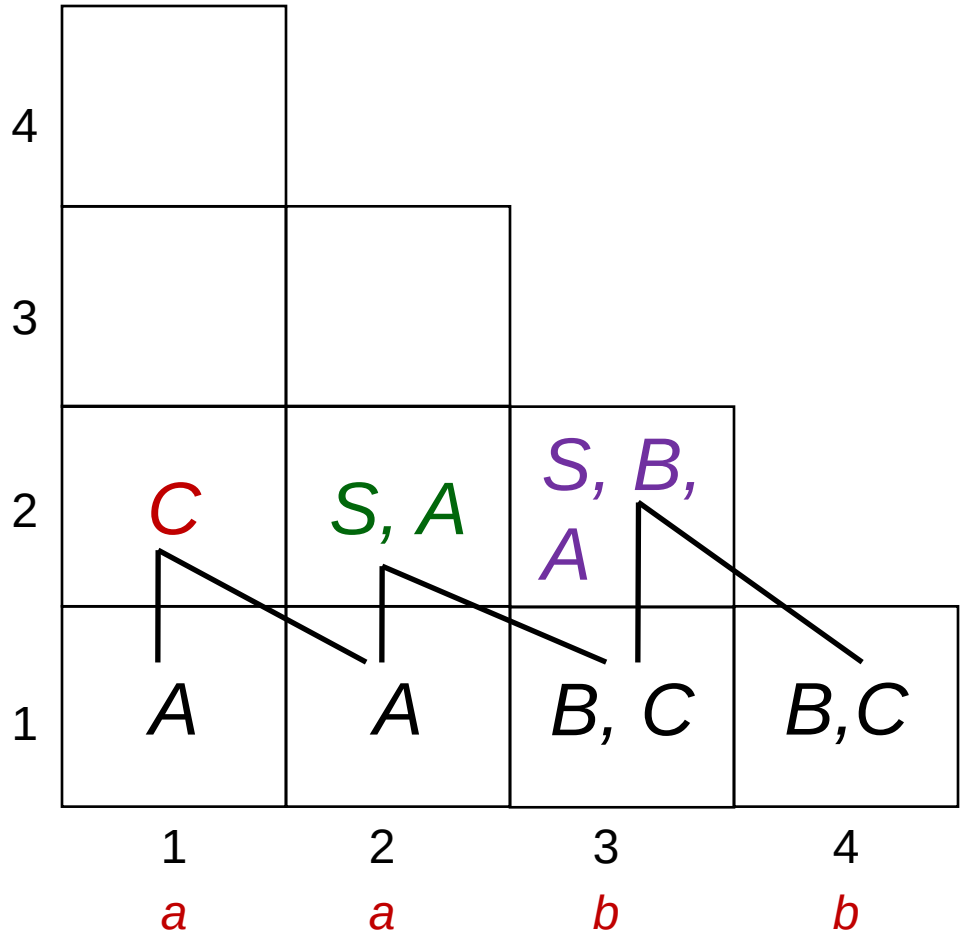
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



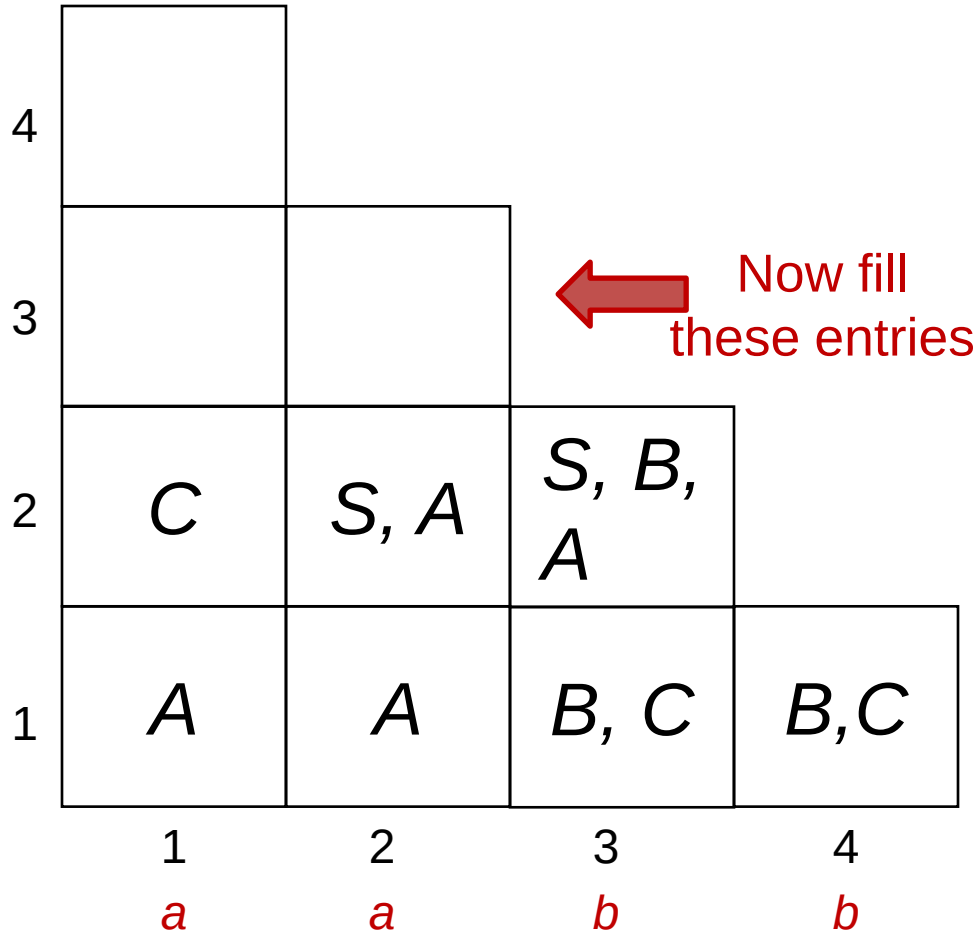
CYK Parsing Algorithm

S -> *AB* | *BB*

A -> *CC* | *AB* | *a*

B -> *BB* |*CA* | *b*,

C -> *BA* | *AA* | *b*



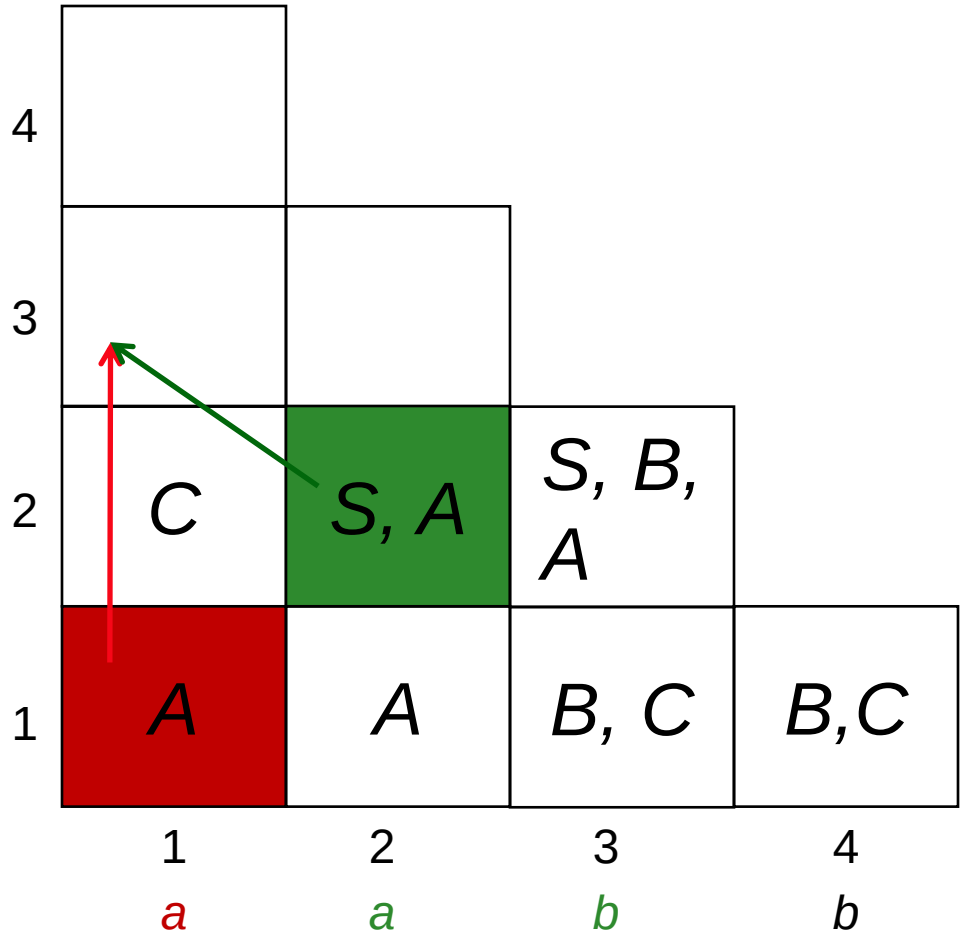
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



Input string to parse

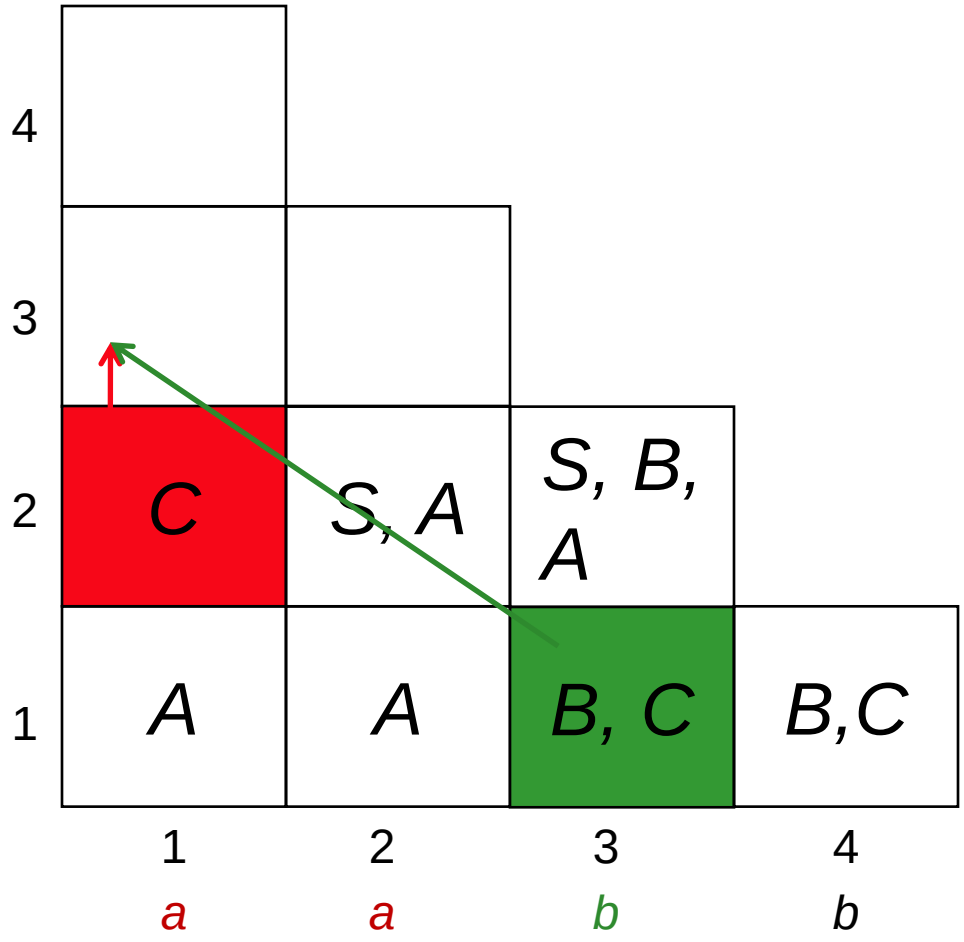
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



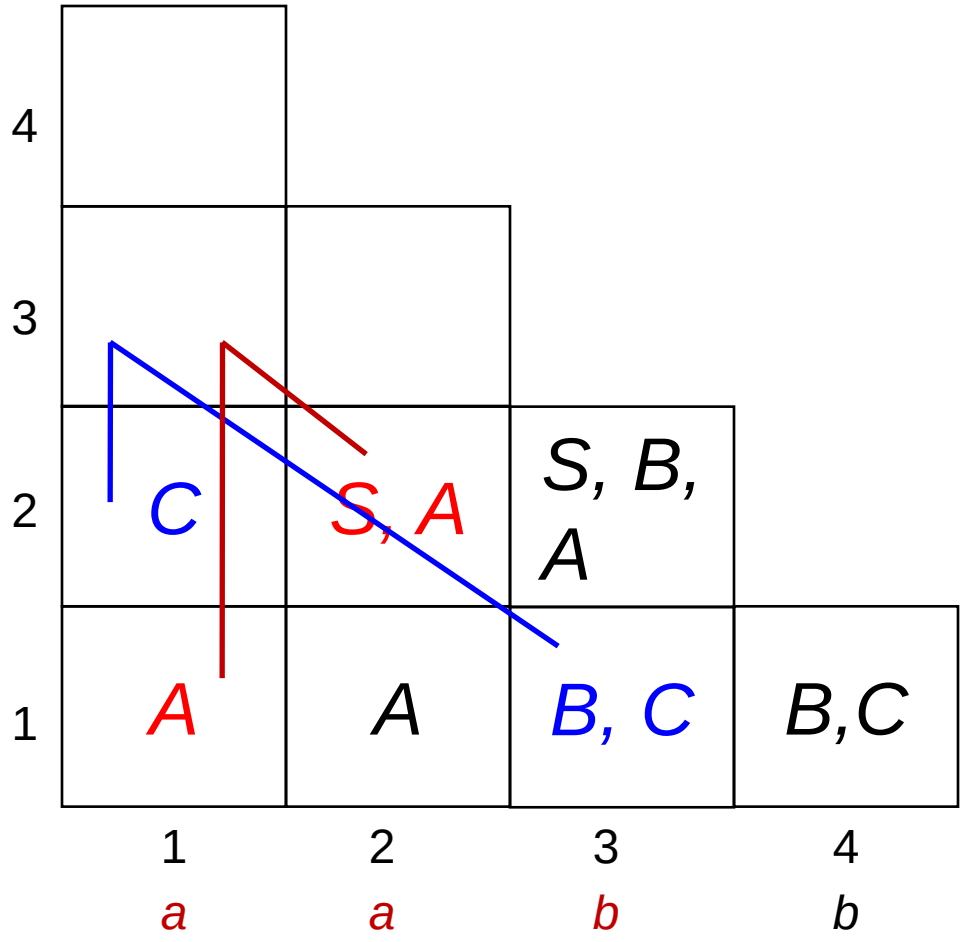
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



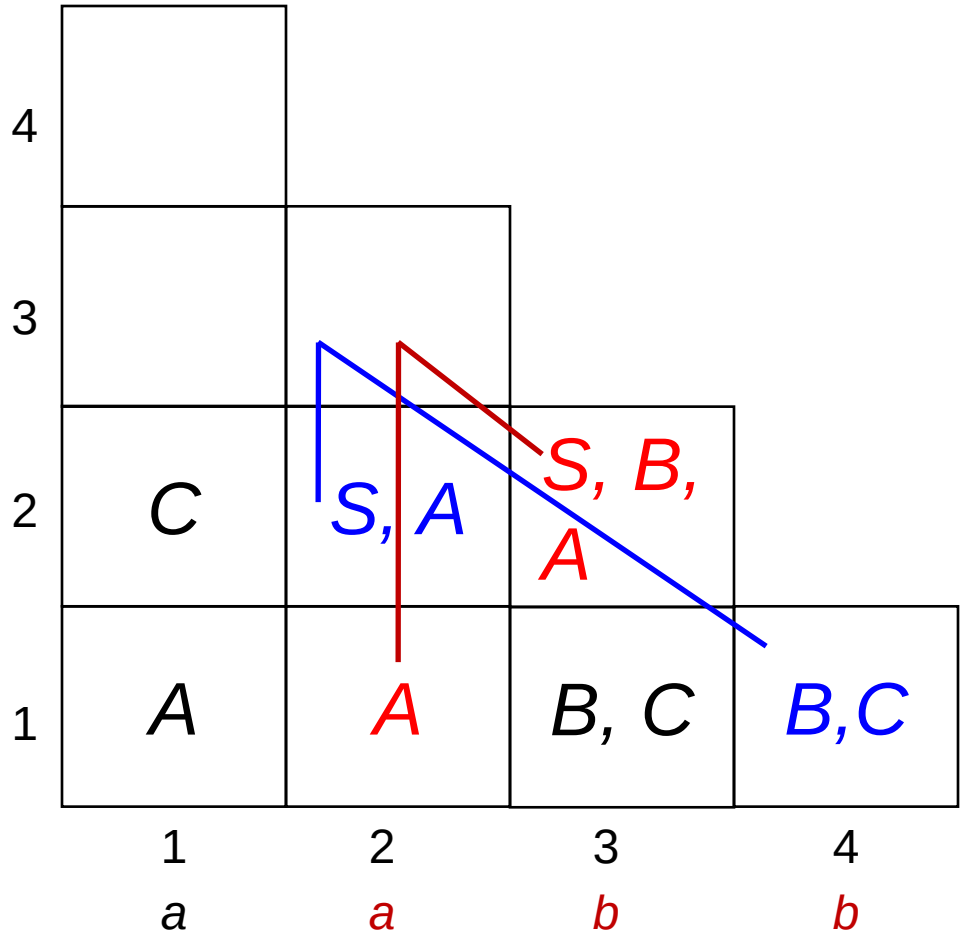
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



Input string to parse

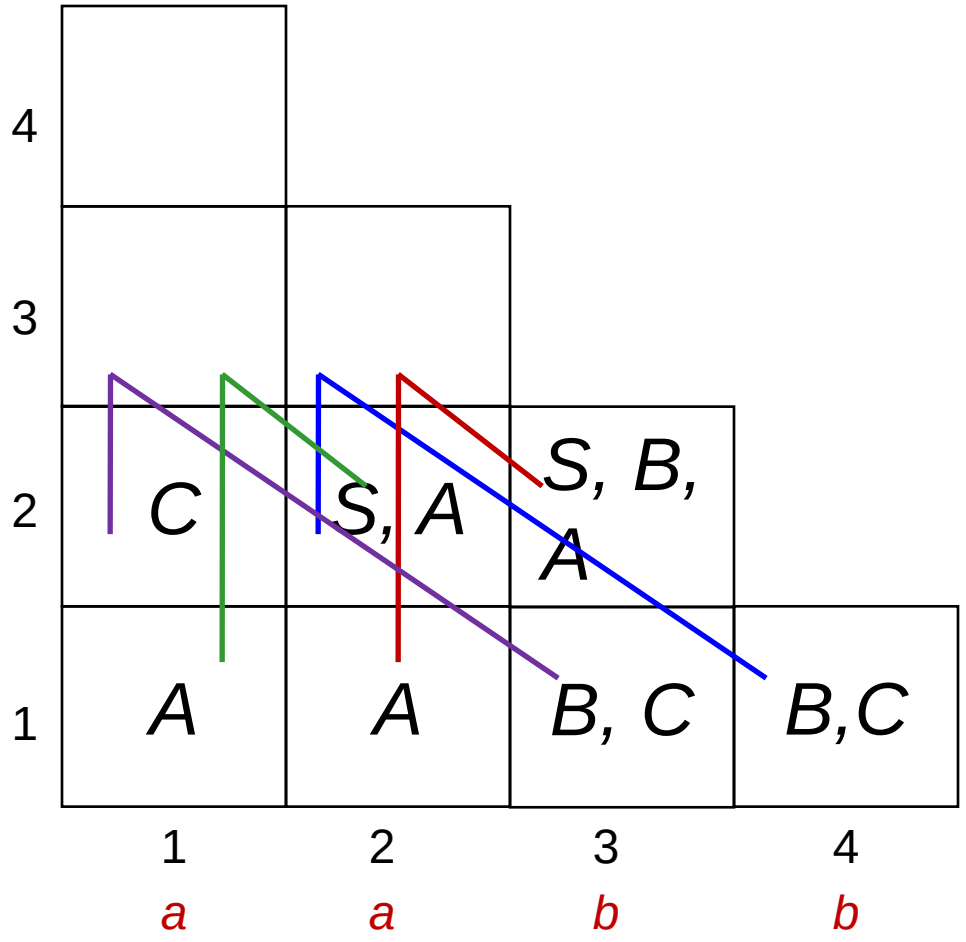
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



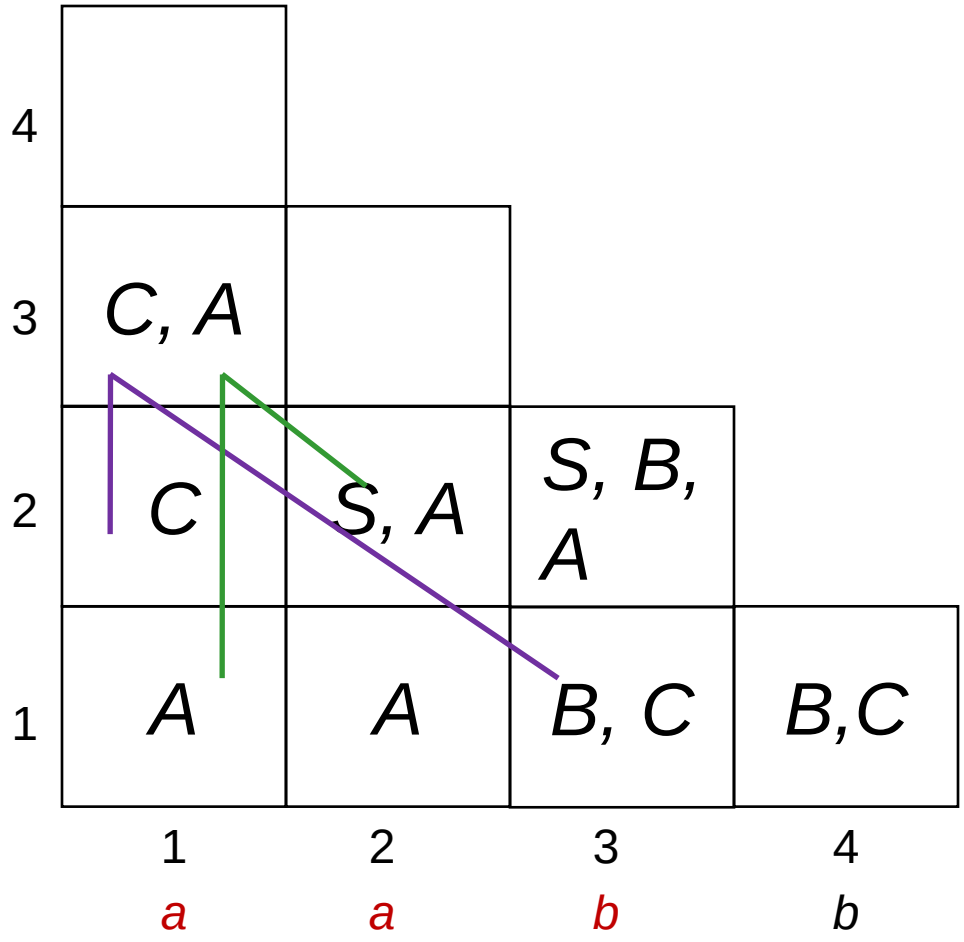
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



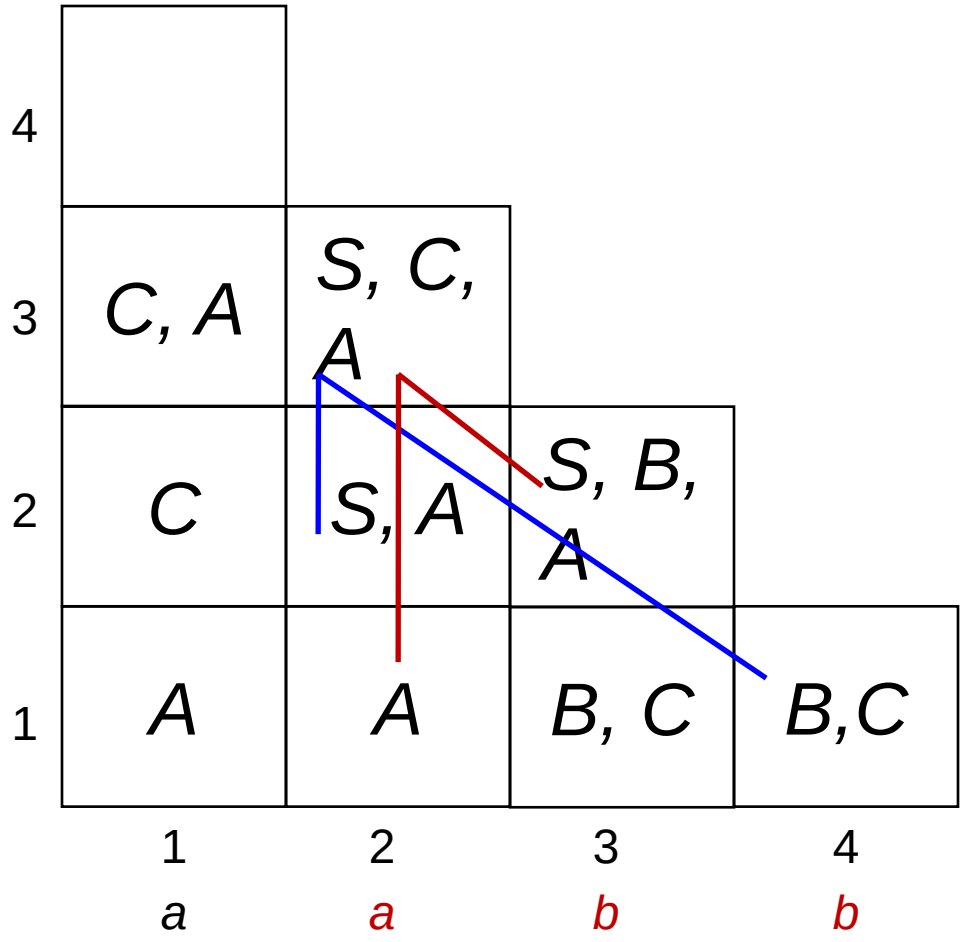
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



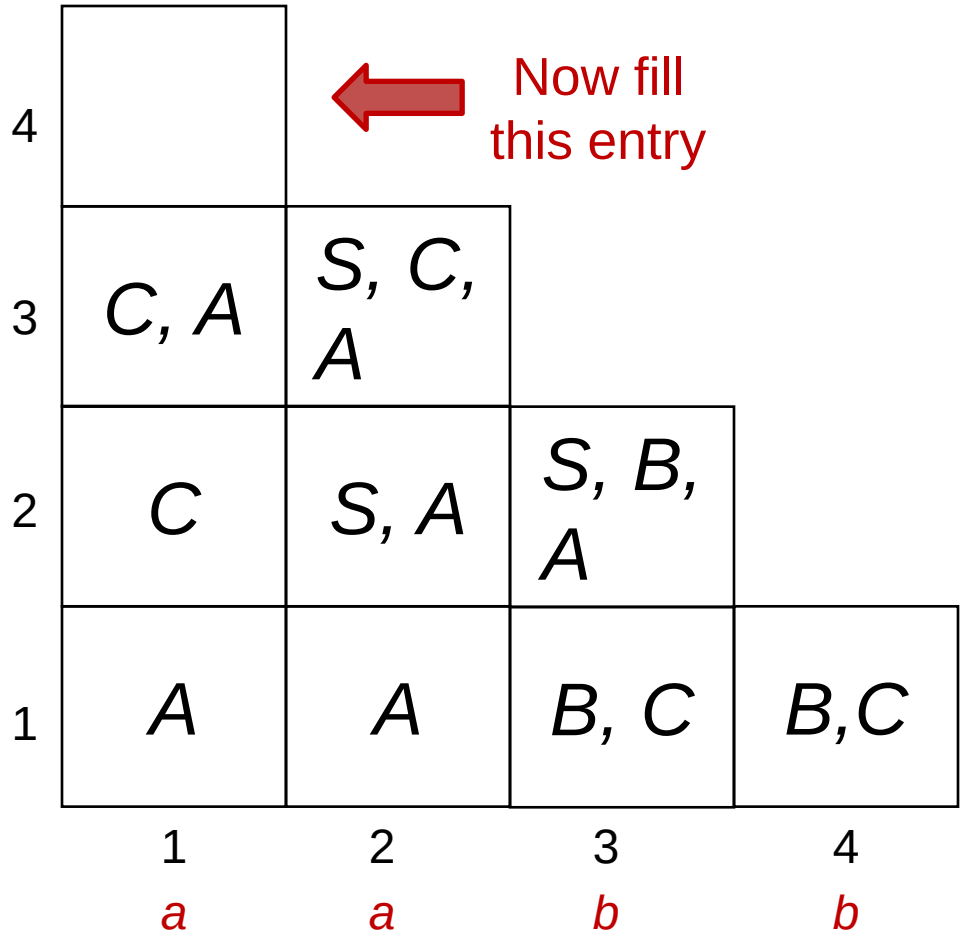
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



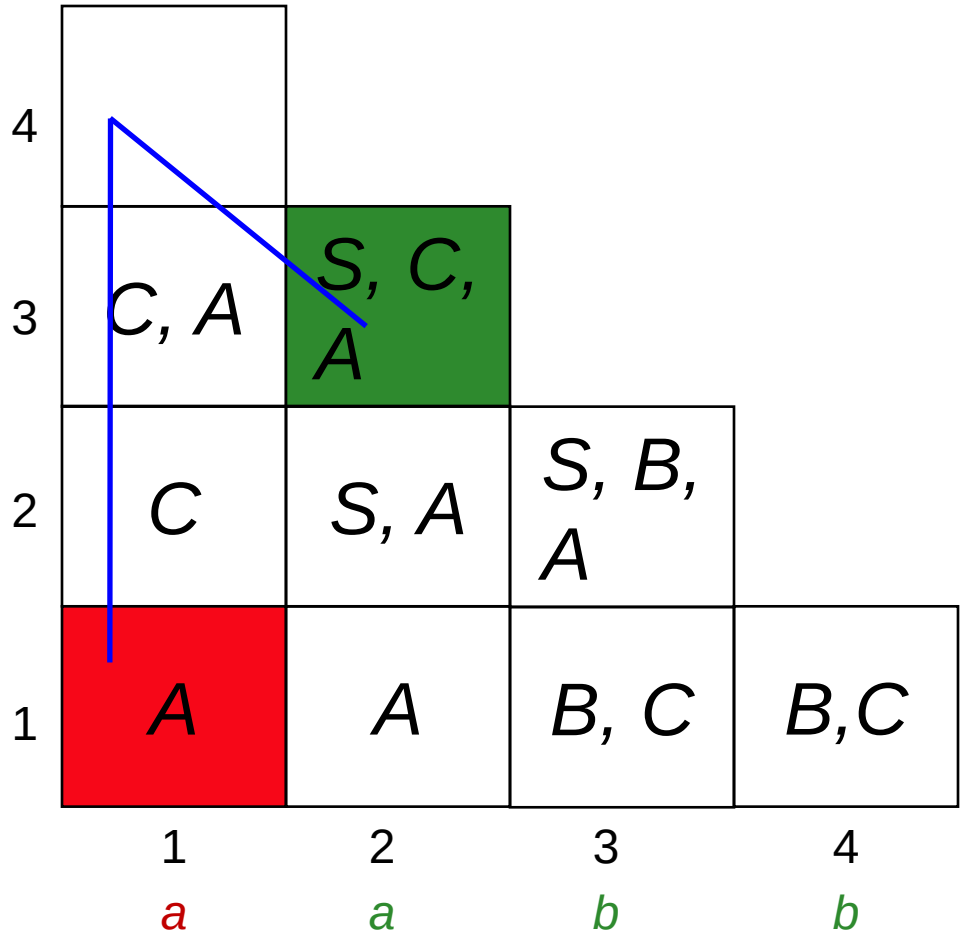
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



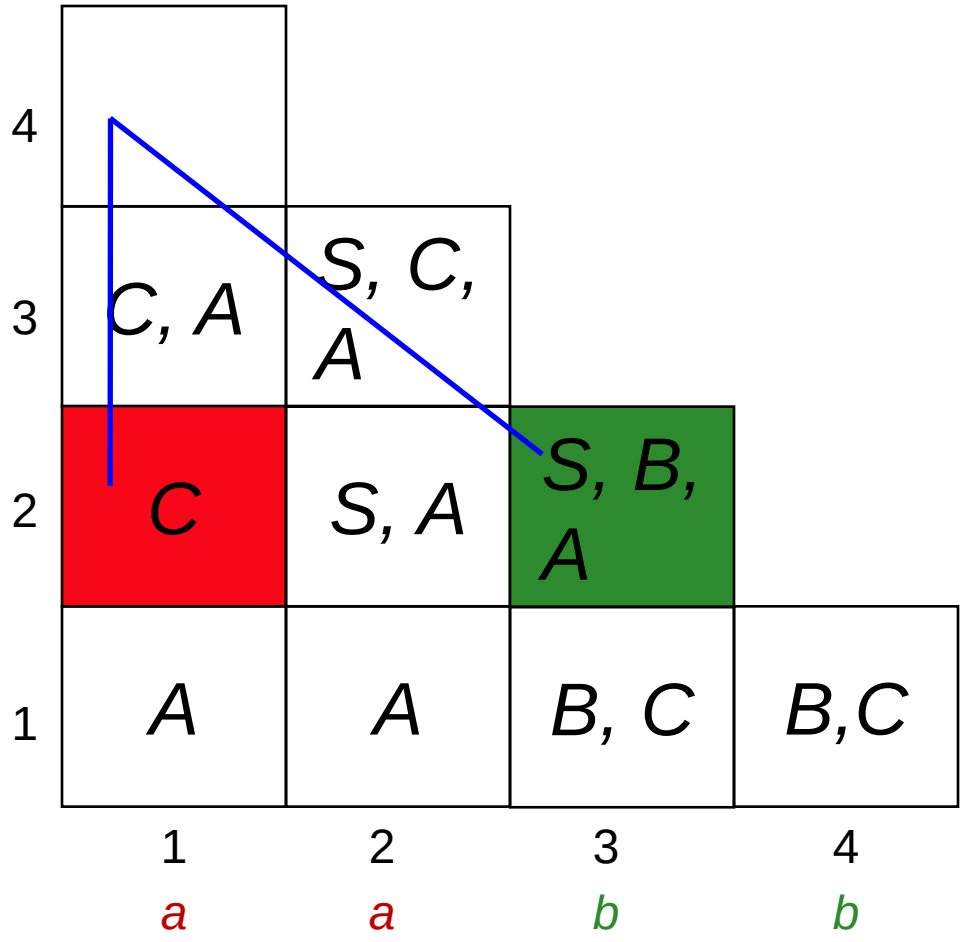
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



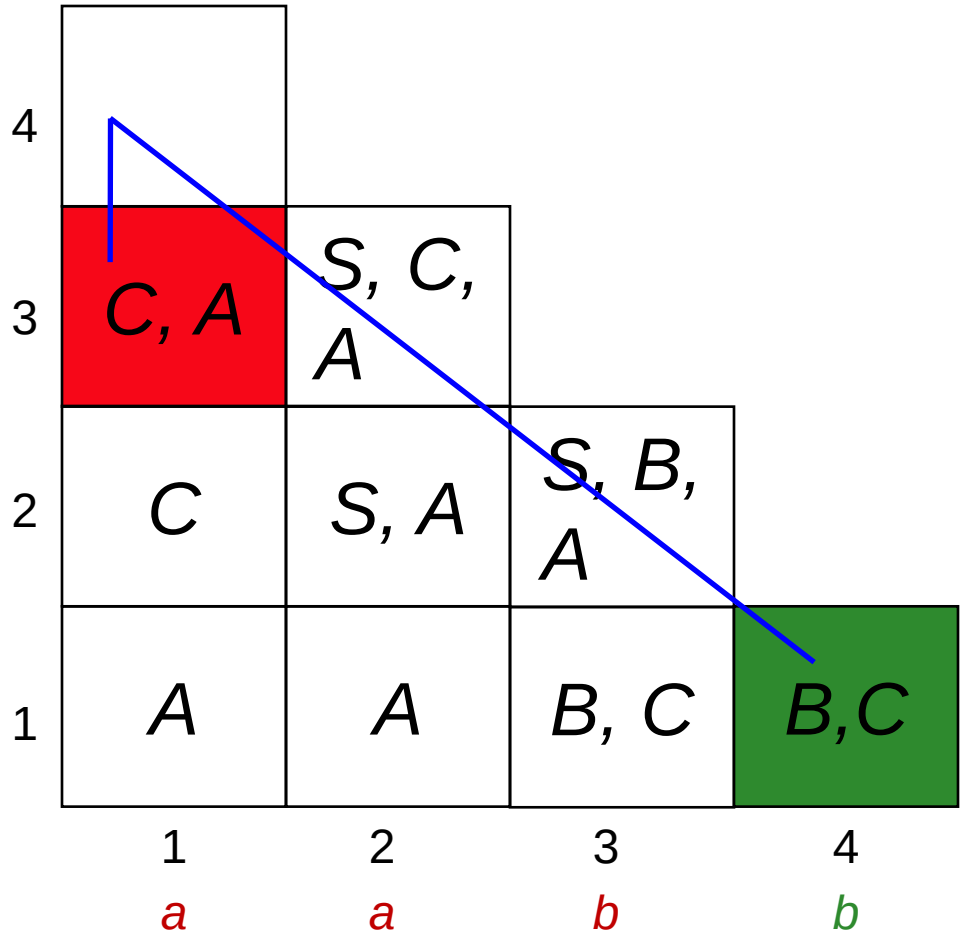
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



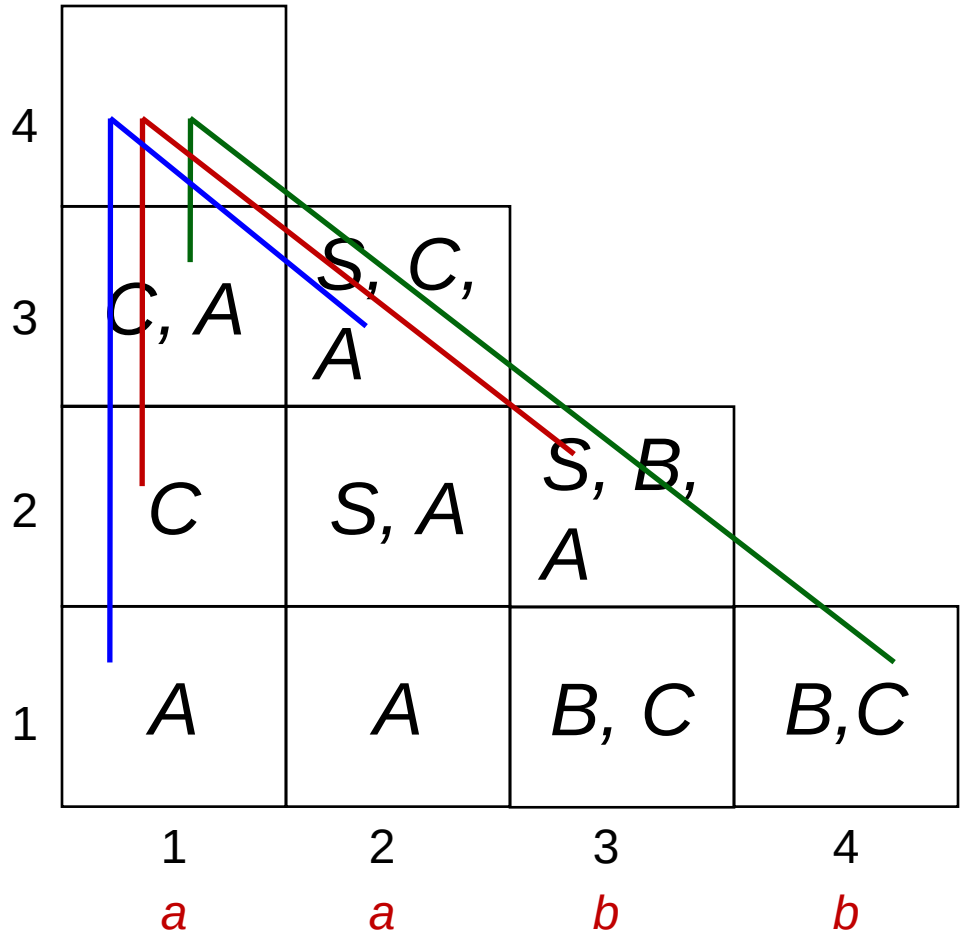
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



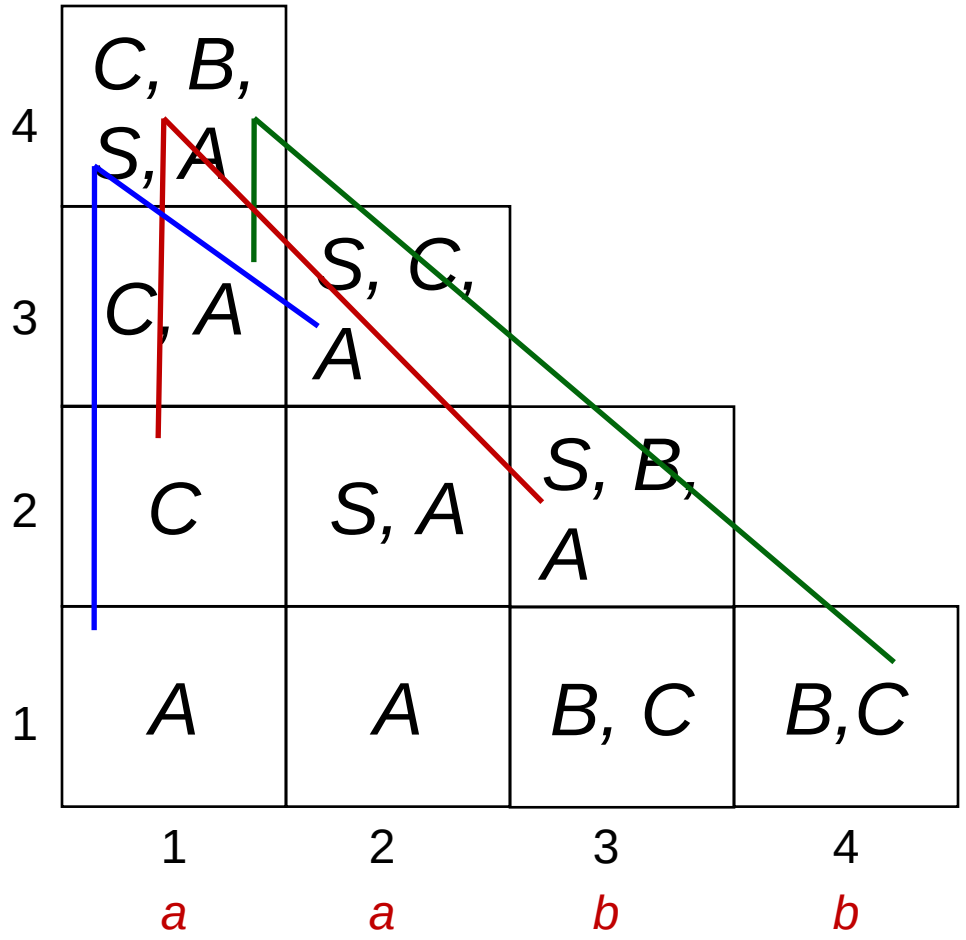
CYK Parsing Algorithm

S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b



CYK Parsing Algorithm

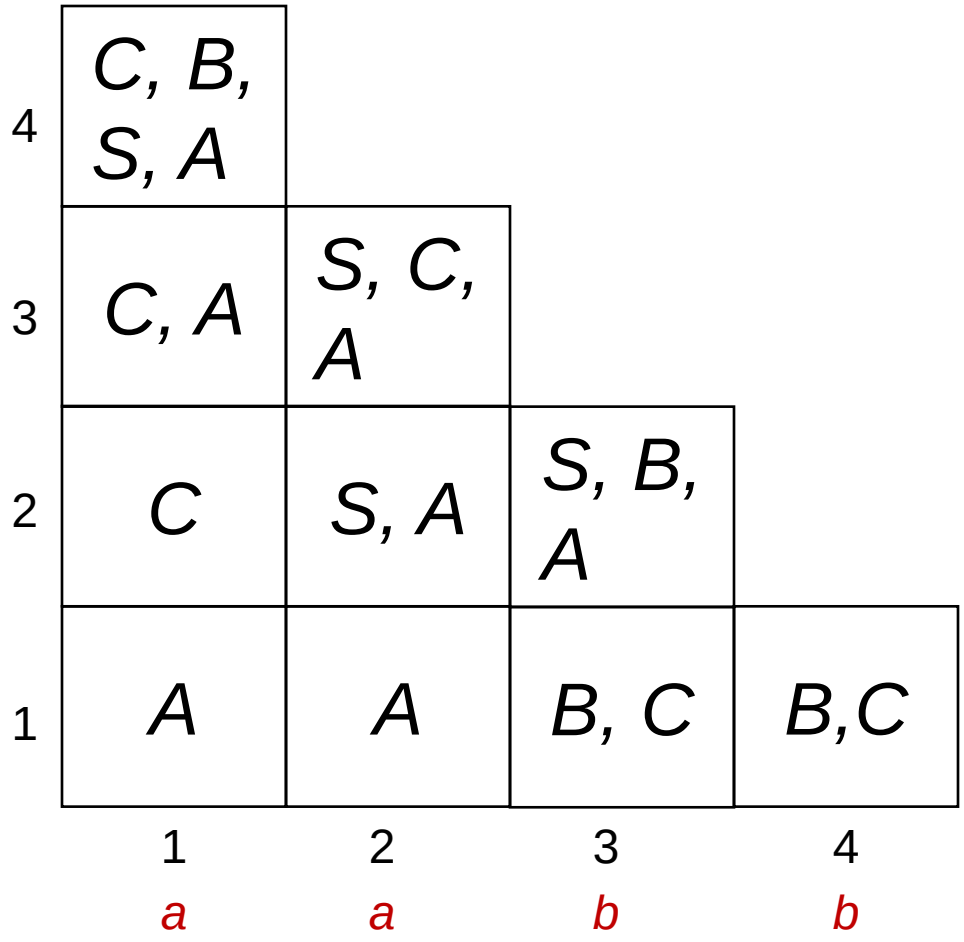
S -> AB | BB

A -> CC | AB | a

B -> BB |CA | b,

C -> BA | AA | b

The string *aabb* is defined in *G*



CYK Parsing Algorithm

- What happens if the productions are not in CNF, e.g.,

  $S \rightarrow bA_1 \mid cA_2$

  $A_1 \rightarrow bA_2$

  $A_2 \rightarrow b \mid aA_2$

CYK Parsing Algorithm

- What happens if the productions are not in CNF, e.g.,

  $S \rightarrow bA_1 \mid cA_2$

  $A_1 \rightarrow bA_2$

  $A_2 \rightarrow b \mid aA_2$

We can convert them into CNF

CYK Parsing Algorithm

- What happens if the productions are not in CNF, e.g.,

$S \rightarrow bA_1 \mid cA_2$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow b \mid aA_2$

Equivalent CNF

$S \rightarrow A_3A_1$

$A_3 \rightarrow b$

CYK Parsing Algorithm

- What happens if the productions are not in CNF, e.g.,

$S \rightarrow bA_1 \mid cA_2$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow b \mid aA_2$

Equivalent CNF

$S \rightarrow A_3A_1$

$A_3 \rightarrow b$

$S \rightarrow A_4A_2$

$A_4 \rightarrow c$

CYK Parsing Algorithm

- What happens if the productions are not in CNF, e.g.,

$S \rightarrow bA_1 \mid cA_2$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow b \mid aA_2$

Equivalent CNF

$S \rightarrow A_3A_1$

$A_3 \rightarrow b$

$S \rightarrow A_4A_2$

$A_4 \rightarrow c$

$A_1 \rightarrow A_3A_2$

$A_2 \rightarrow b$

$A_2 \rightarrow A_5A_2$

$A_5 \rightarrow a$

CYK Parsing Algorithm

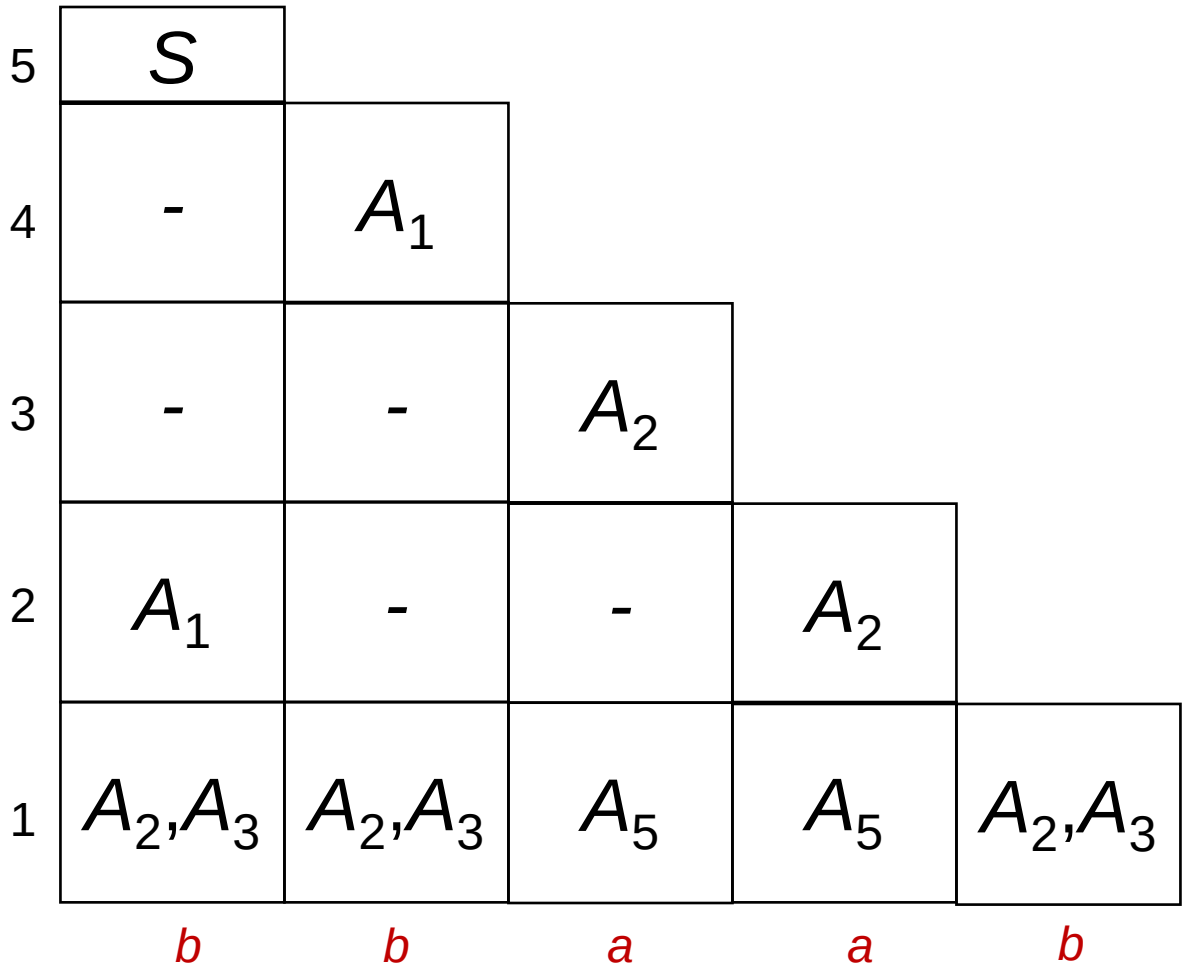
- Example 2: find whether *bbaab* is defined in *G*

$S \rightarrow A_3 A_1$
$A_3 \rightarrow b$
$S \rightarrow A_4 A_2$
$A_4 \rightarrow c$
$A_1 \rightarrow A_3 A_2$
$A_2 \rightarrow b$
$A_2 \rightarrow A_5 A_2$
$A_5 \rightarrow a$

CYK Parsing Algorithm

- Example 2: find whether *bbaab* is defined in *G*

$S \rightarrow A_3A_1$
$A_3 \rightarrow b$
$S \rightarrow A_4A_2$
$A_4 \rightarrow c$
$A_1 \rightarrow A_3A_2$
$A_2 \rightarrow b$
$A_2 \rightarrow A_5A_2$
$A_5 \rightarrow a$



CYK Parsing Algorithm

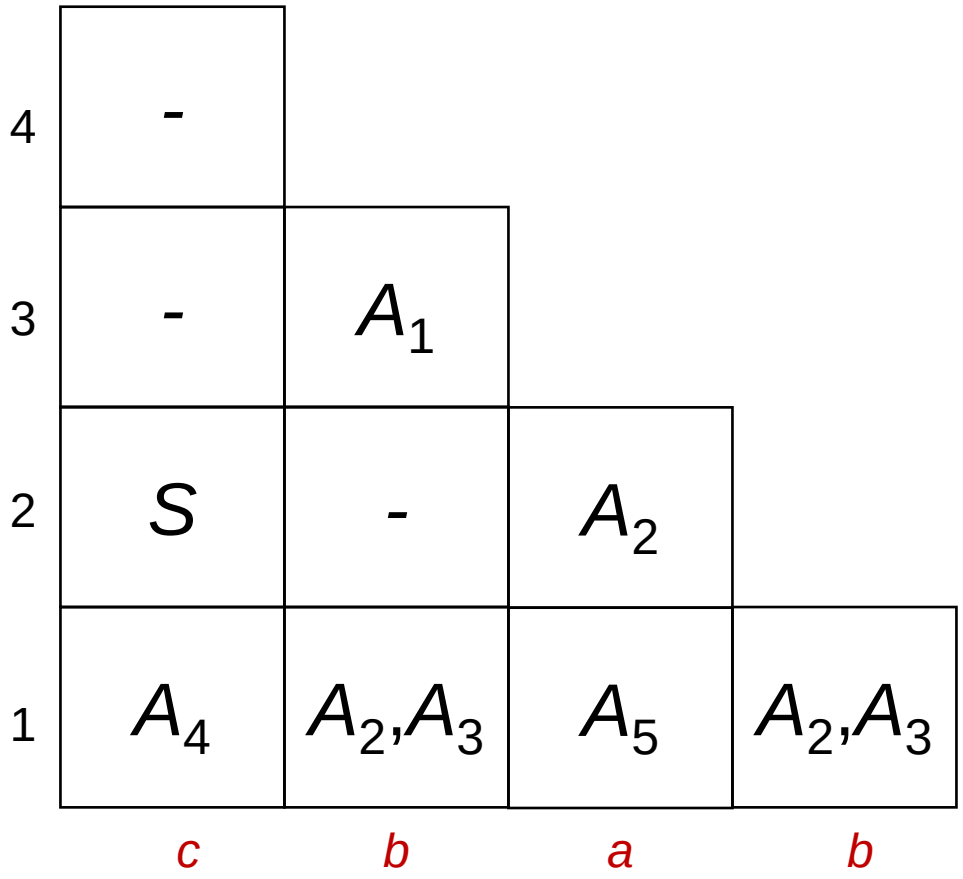
- Example 2: find whether *cbab* is defined in *G*

$S \rightarrow A_3A_1$
$A_3 \rightarrow b$
$S \rightarrow A_4A_2$
$A_4 \rightarrow c$
$A_1 \rightarrow A_3A_2$
$A_2 \rightarrow b$
$A_2 \rightarrow A_5A_2$
$A_5 \rightarrow a$

CYK Parsing Algorithm

- Example 2: find whether *cbab* is defined in *G*

$S \rightarrow A_3 A_1$
$A_3 \rightarrow b$
$S \rightarrow A_4 A_2$
$A_4 \rightarrow c$
$A_1 \rightarrow A_3 A_2$
$A_2 \rightarrow b$
$A_2 \rightarrow A_5 A_2$
$A_5 \rightarrow a$

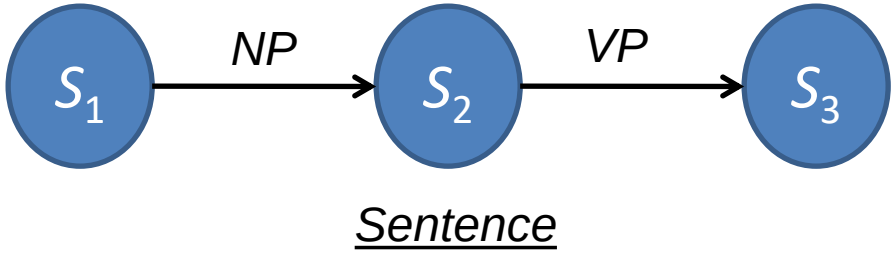


Parsing using Transition Network (TN)

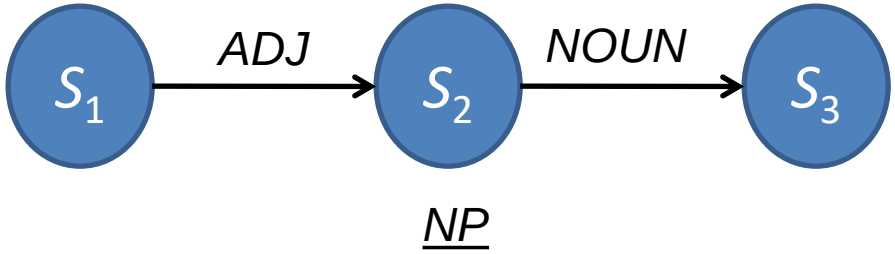
- Consists of nodes and arc
- Nodes represent states
- Arcs are labeled with terminal or non-terminal

An Example of Transition Network (TN)

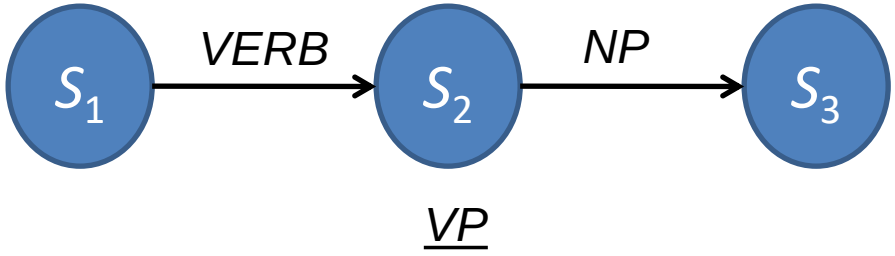
SENTENCE -> NP + VP



NP -> ADJ + NOUN

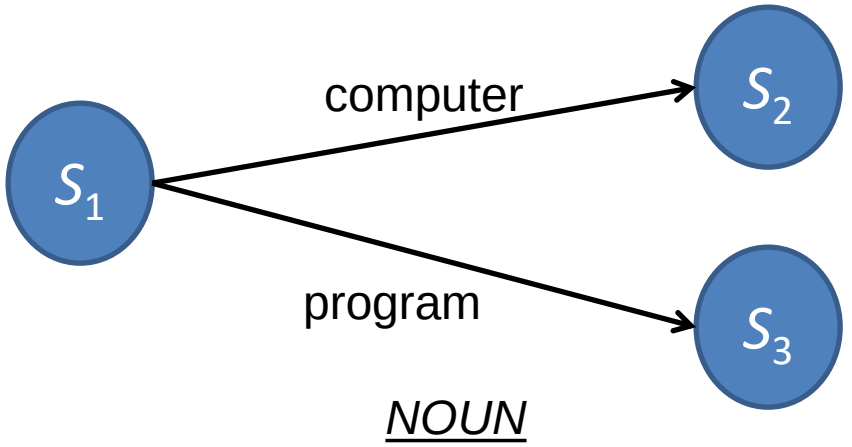


VP -> VERB + NP

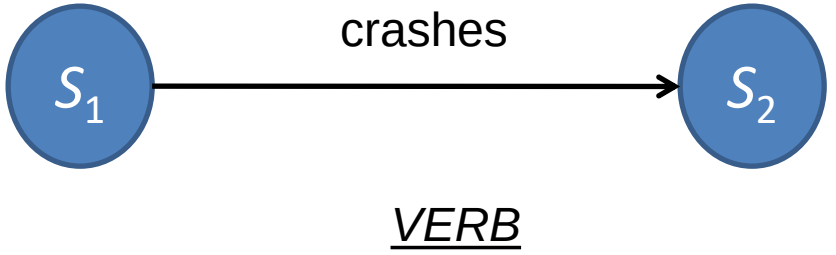


An Example of Transition Network (TN)

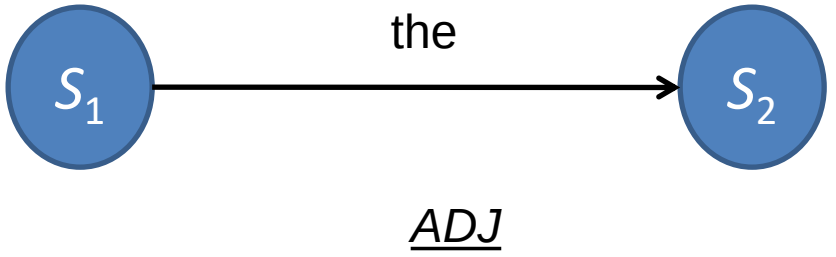
NOUN -> computer | program



VERB -> crashes

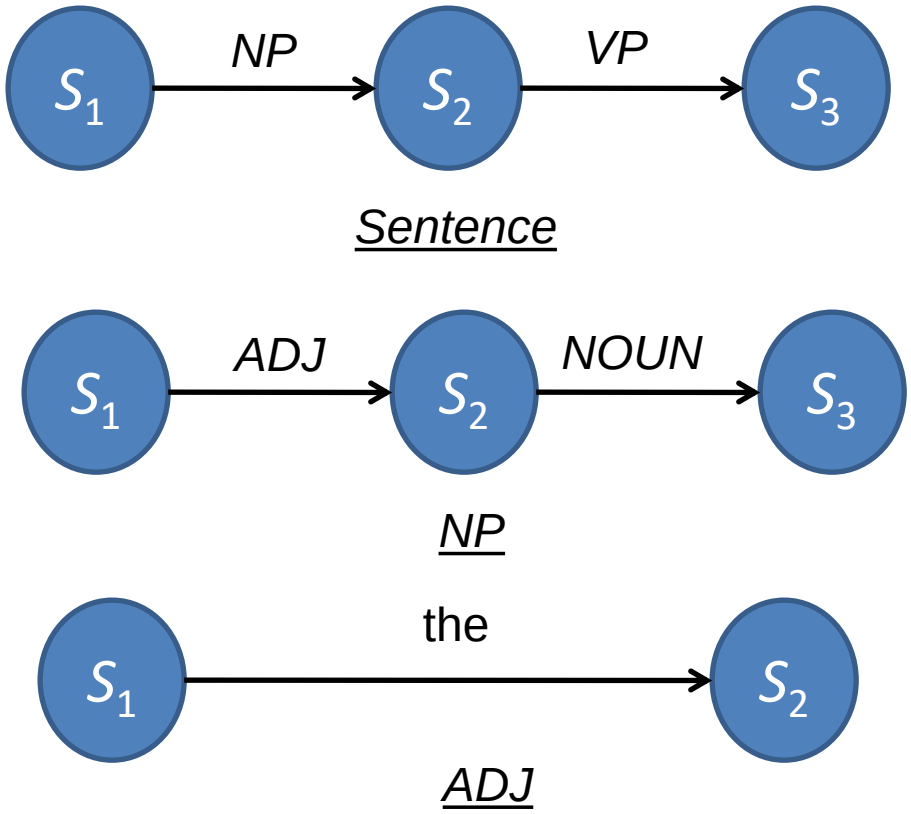


ADJ -> the



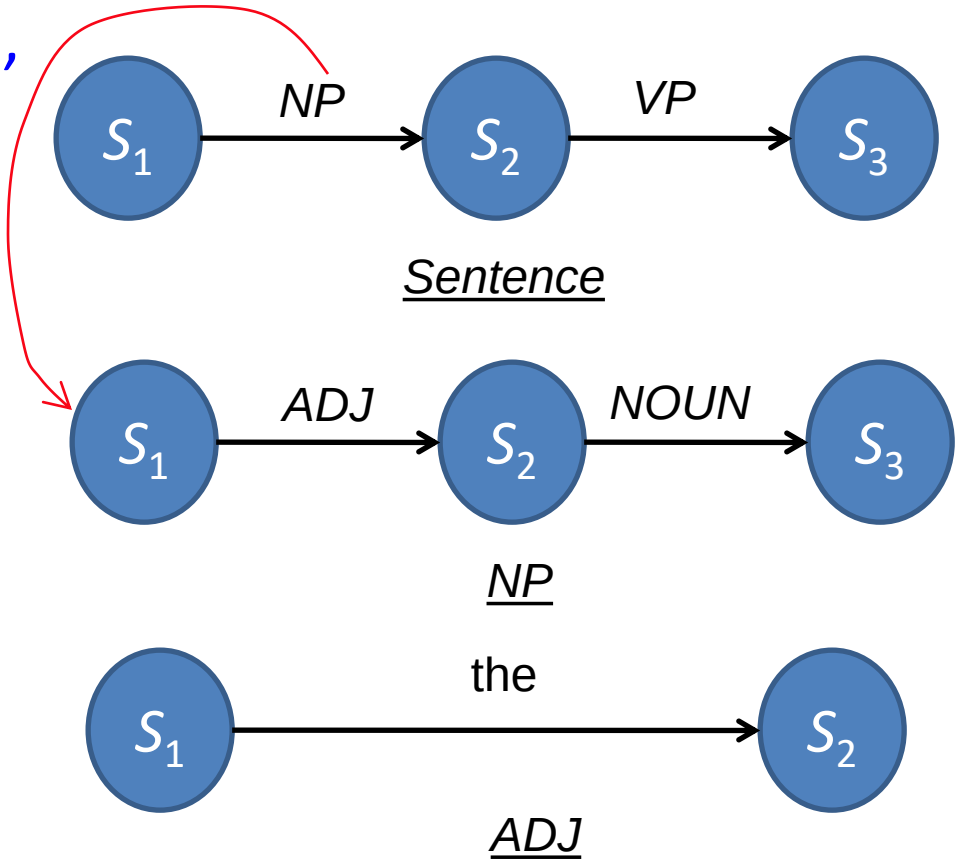
Parsing in TN

- Starts at an initial node
- Sequentially checks input string against the arc label



Parsing in TN

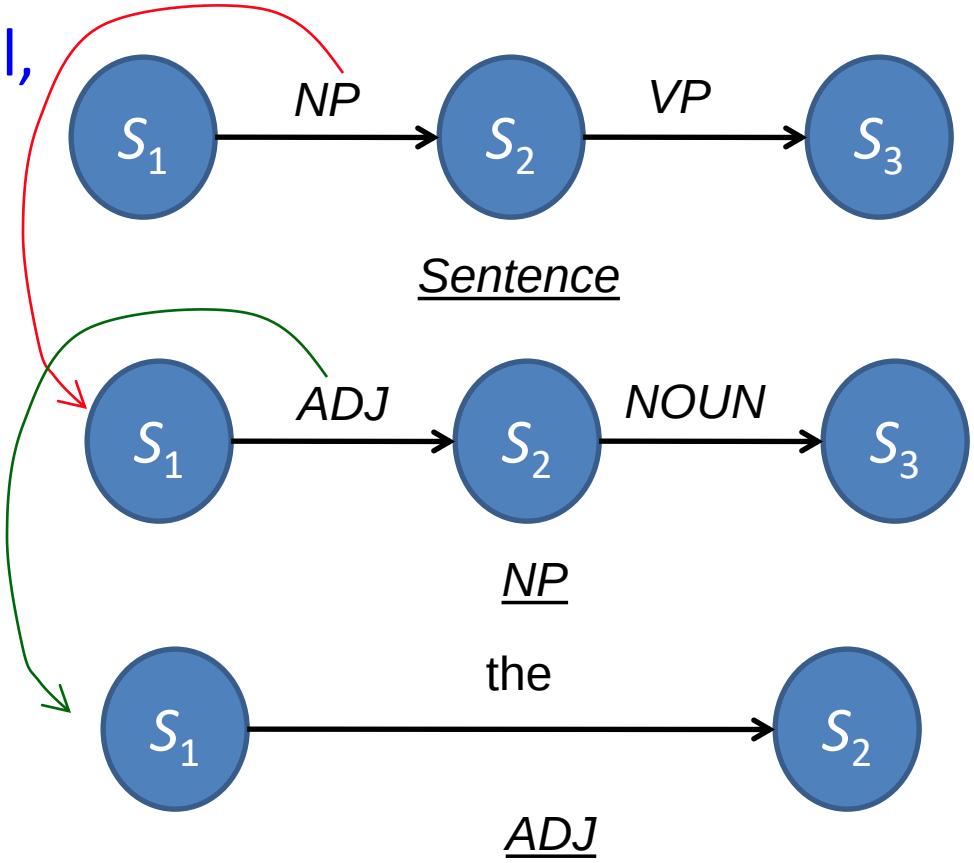
- Starts at an initial node
- Sequentially checks input string against the arc label
- If arc label is non-terminal, control passes to another TN



Parsing in TN

- Starts at an initial node
- Sequentially checks input string against the arc label
- If arc label is non-terminal, control passes to another TN

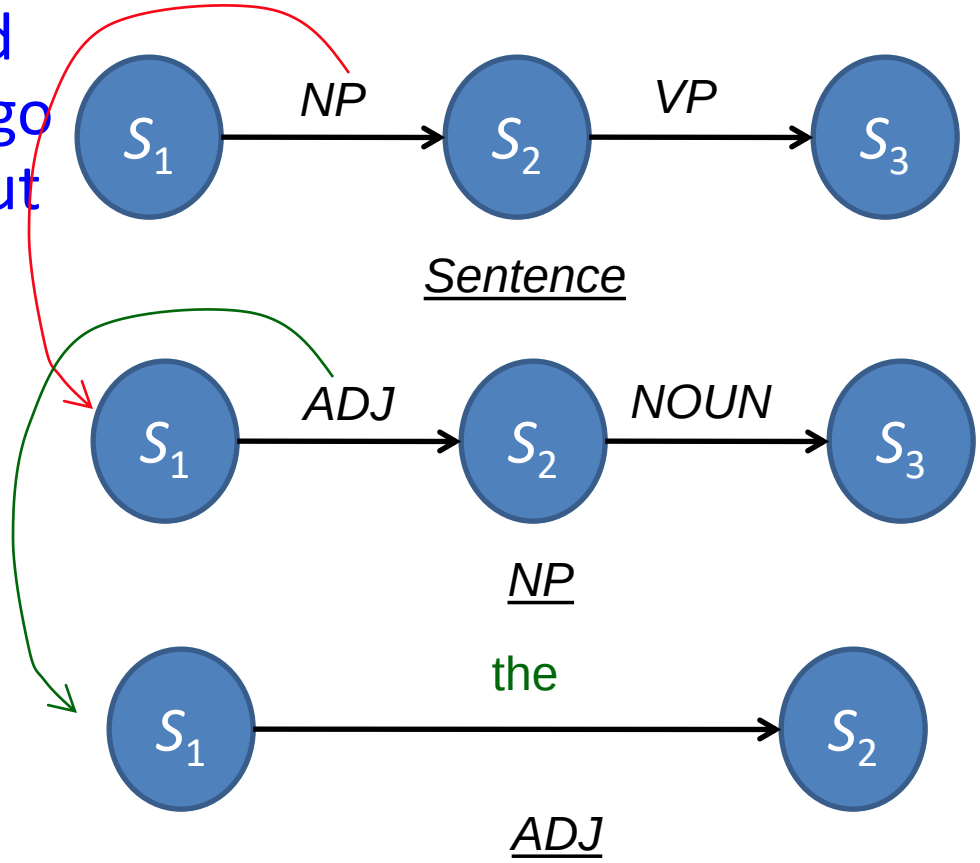
the program crashes the computer



Parsing in TN

- Starts at an initial node
- Sequentially checks input string against the arc label
- If arc label is terminal and matches with the string, go to next node and the input is consumed

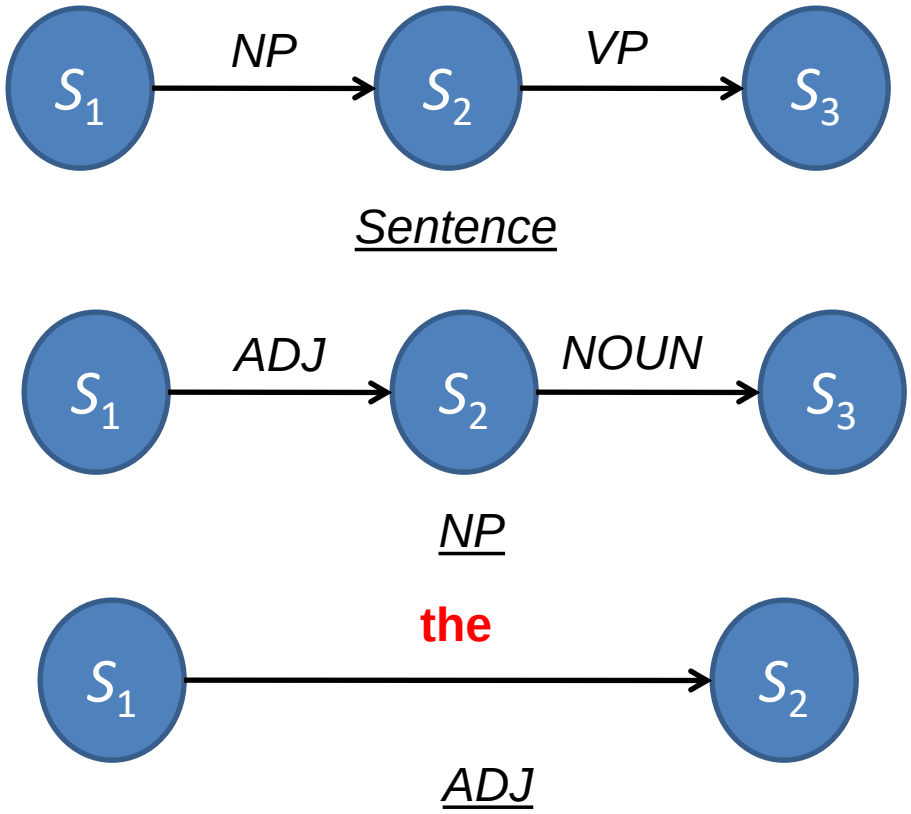
the program crashes the computer



Parsing in TN

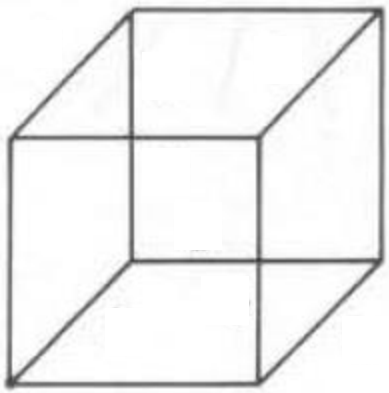
- Starts at an initial node
- Sequentially checks input string against the arc label
- If arc label is terminal and doesn't match with the string, either backtrack or indicate a fail

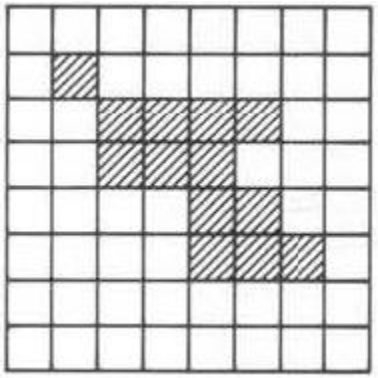
a program crashes the computer



Capability of String Grammar

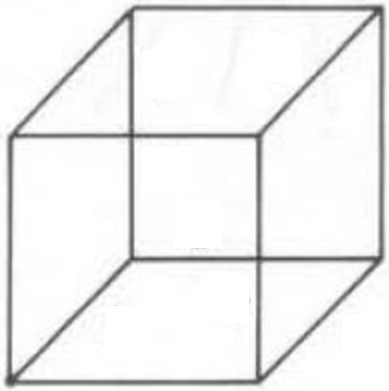
- able to represent 1D pattern
- difficult to classify patterns like these:

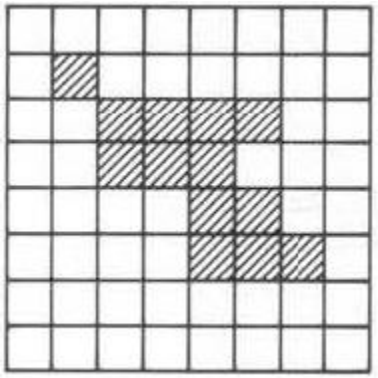




Capability of String Grammar

- These structure contains
  - information in both direction
  - hierarchical structure

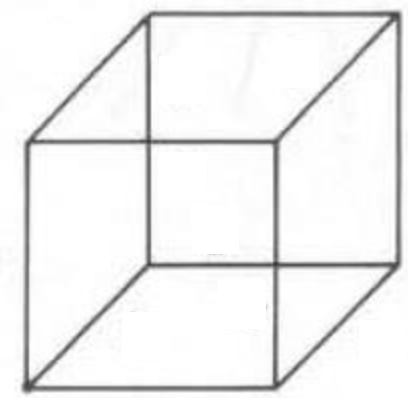


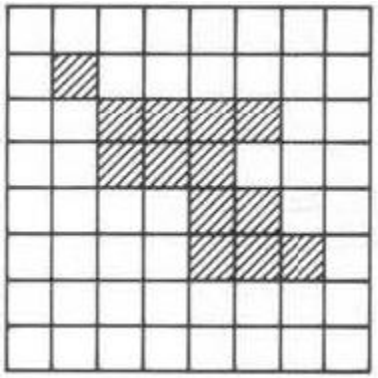


Capability of String Grammar

- These structure contains
  - information in both direction
  - hierarchical structure

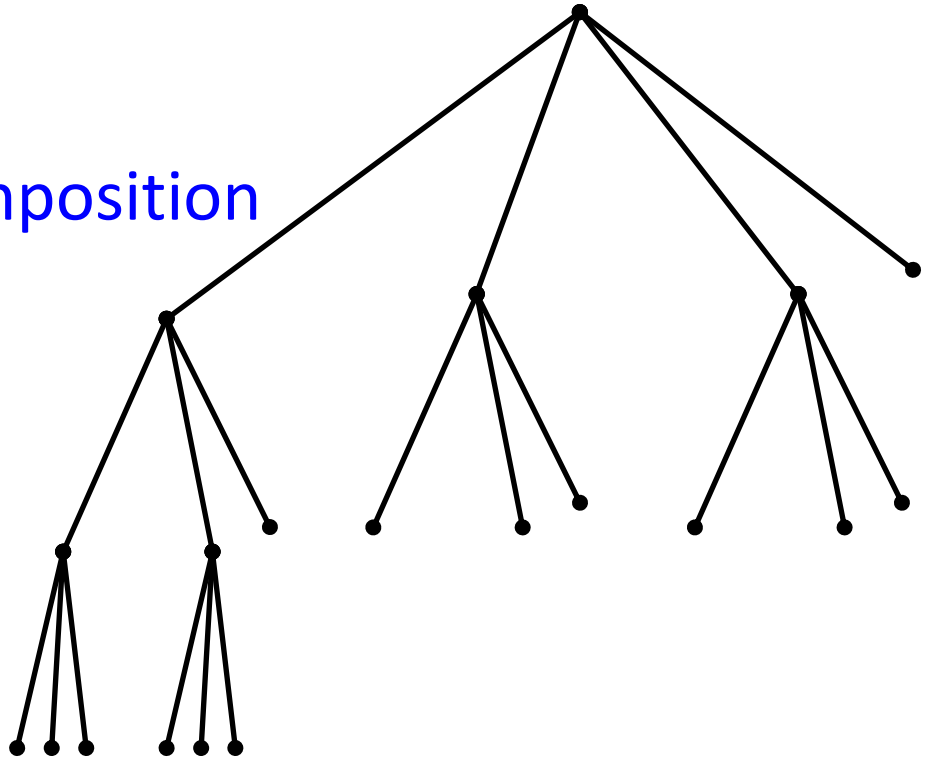
Solution: Higher Order Grammar



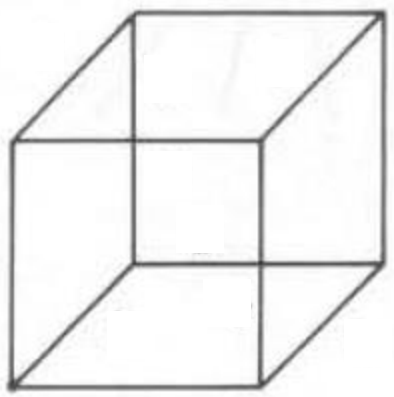


Higher Order Grammar: Tree Grammar

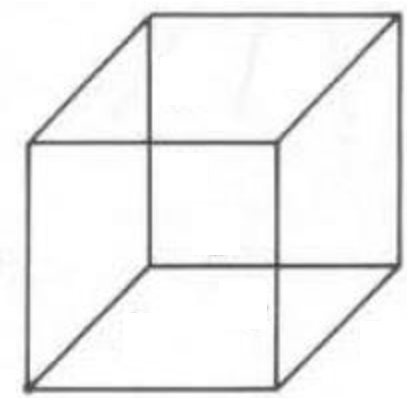
- Information is stored in
  - *nodes*: as primitives or sub-structures
  - *edges*: relation between primitives or sub-structures
- Allows hierarchical decomposition of a complex structure

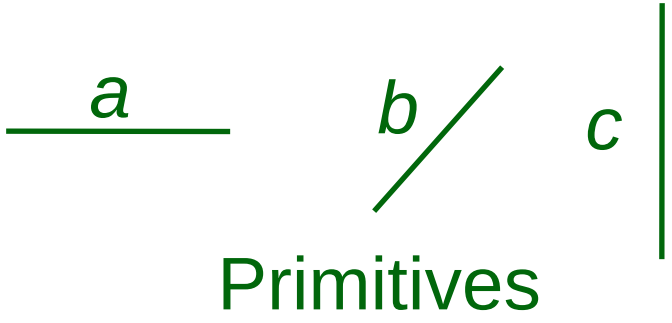


Representation using Tree (1)

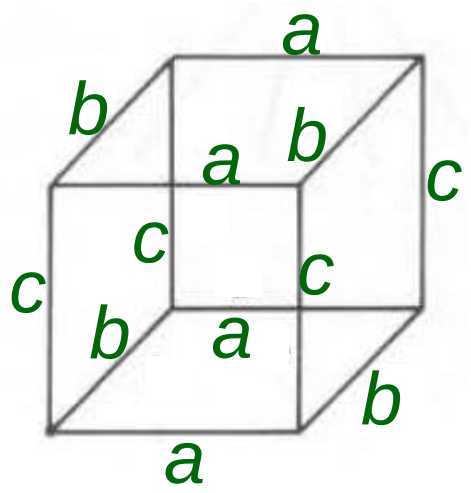


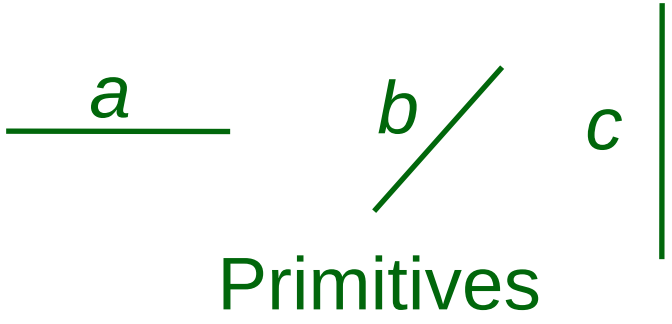
Representation using Tree (1)





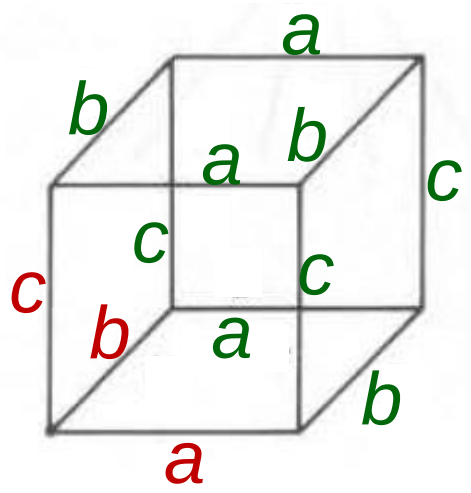
Representation using Tree (1)

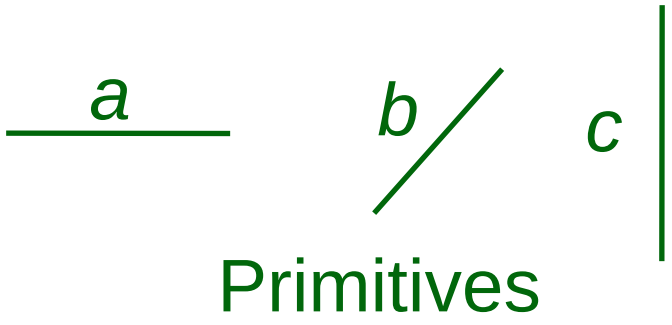


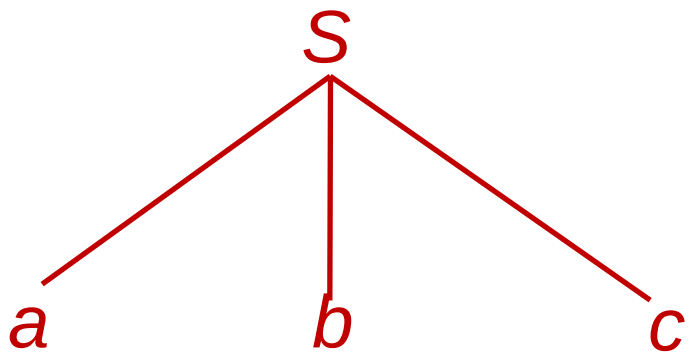


Primitives

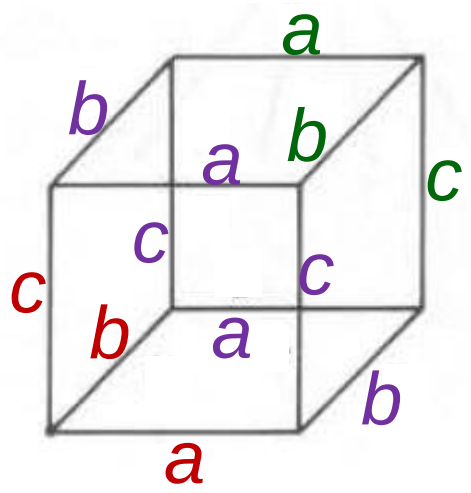
Representation using Tree (1)

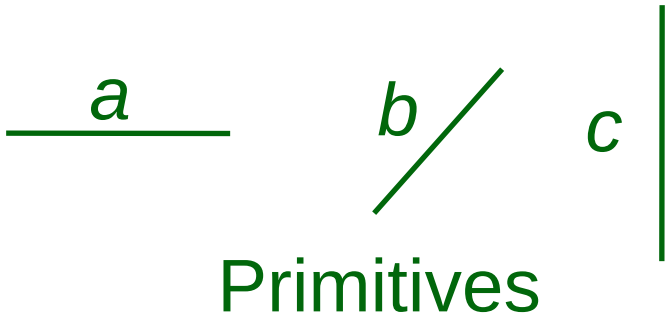


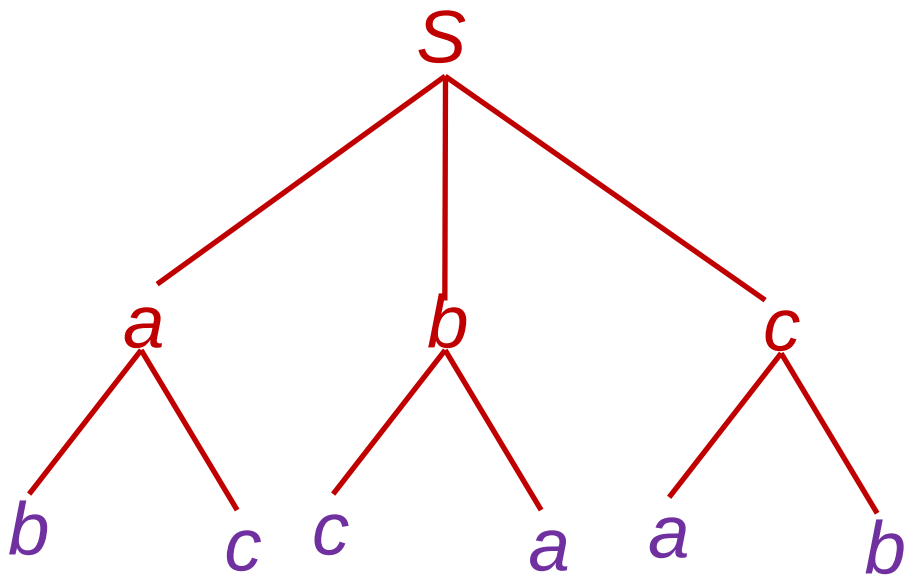




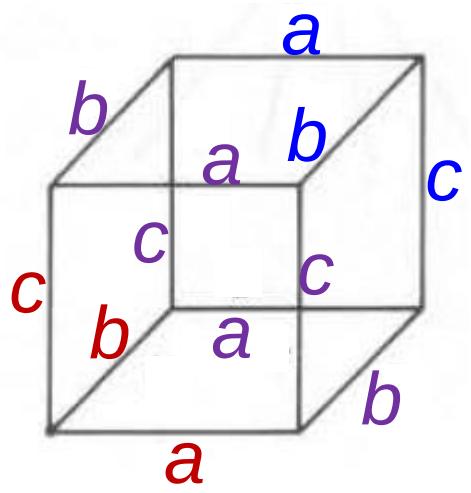
Representation using Tree (1)

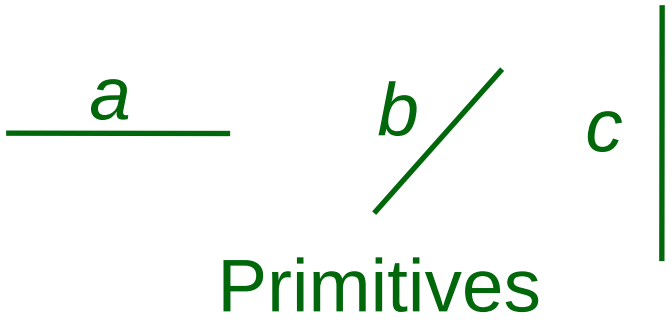


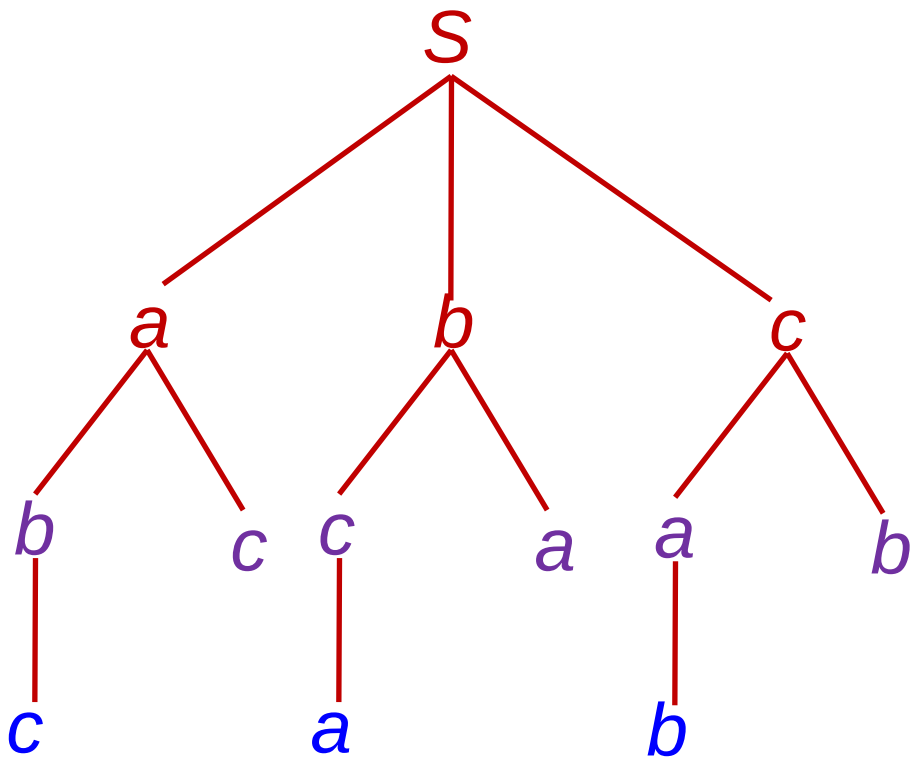




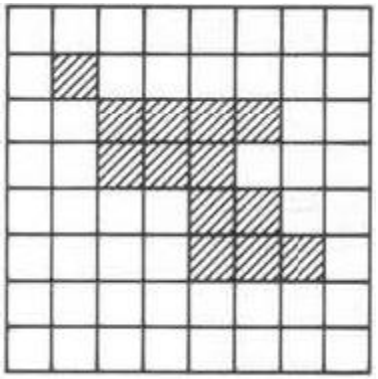
Representation using Tree (1)



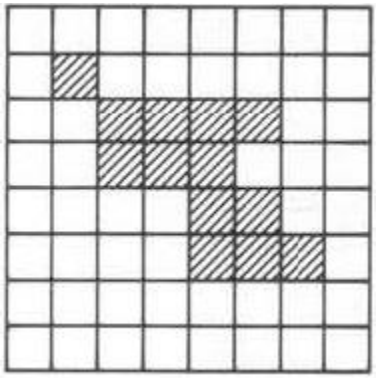




Representation using Tree (2)



Representation using Tree (2)



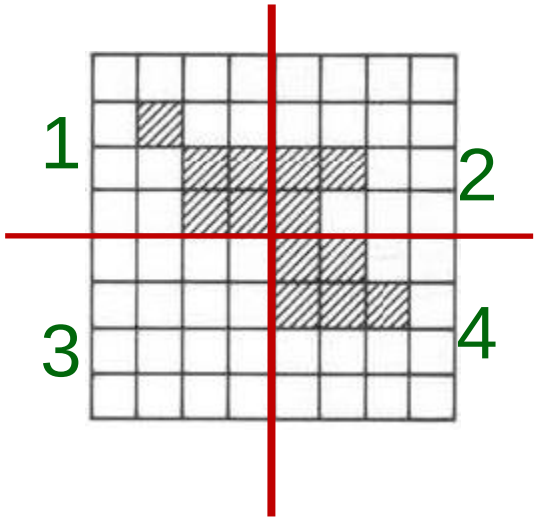
Primitives

Black region

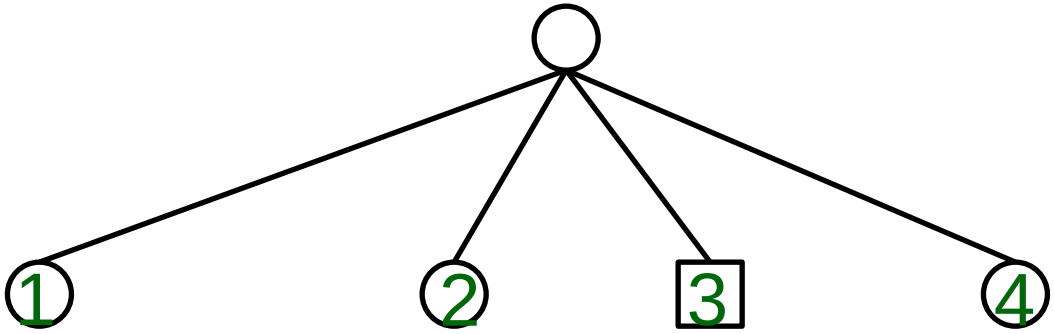
White region

Gray region

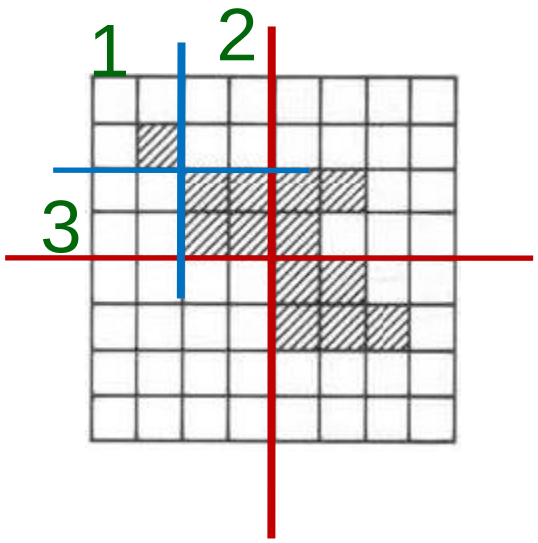
Representation using Tree (2)



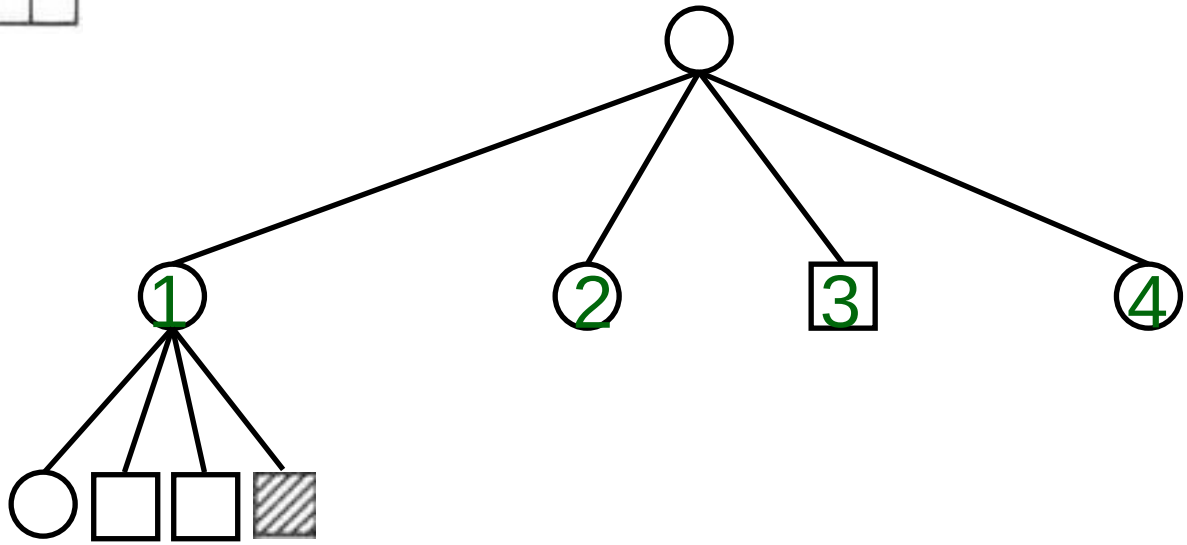
Primitives



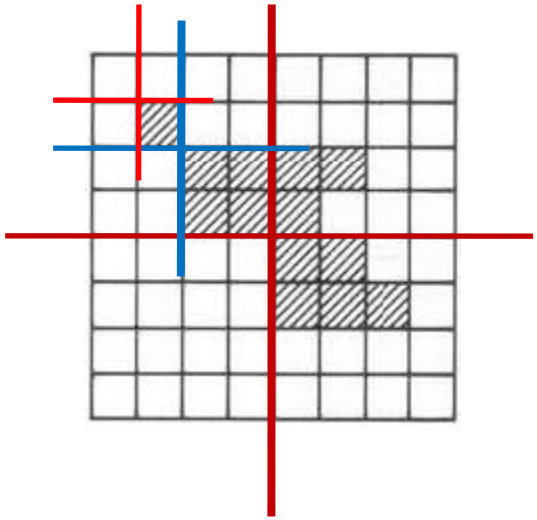
Representation using Tree (2)



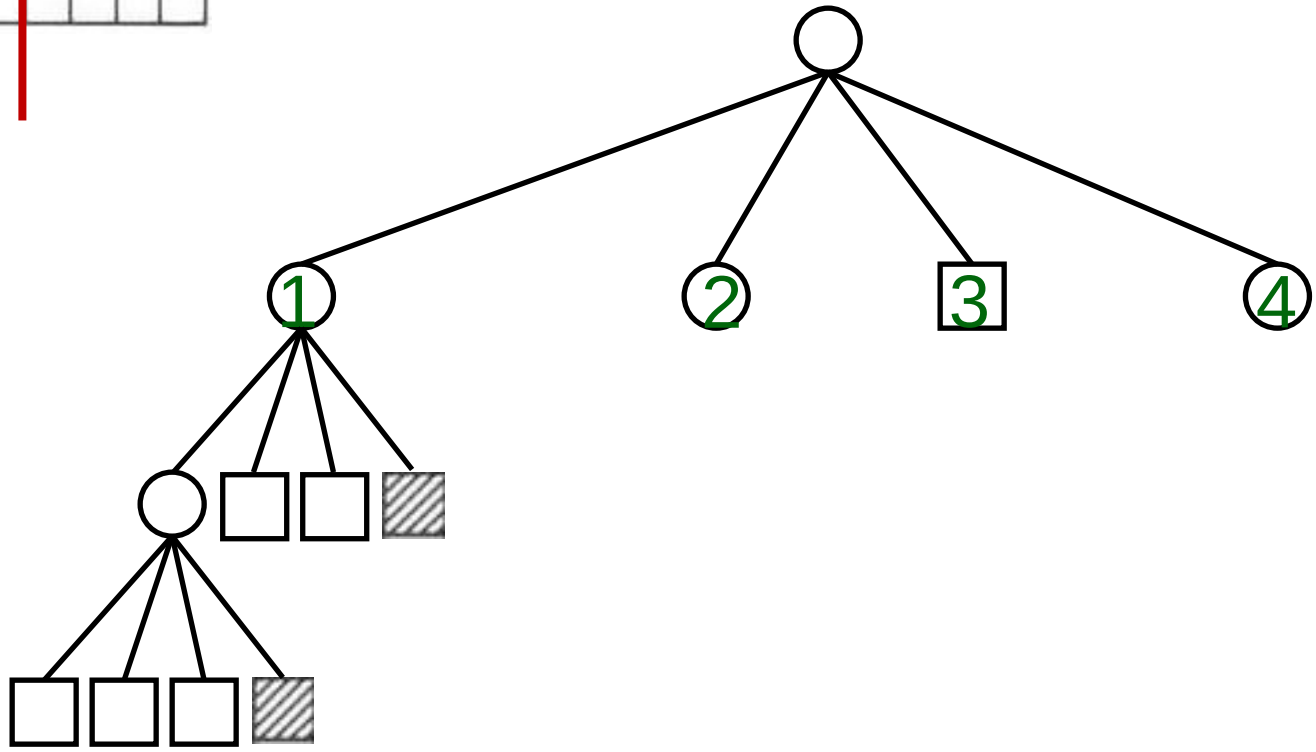
Primitives



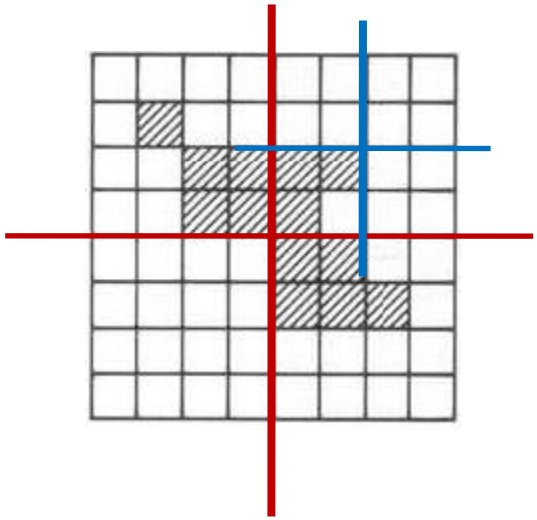
Representation using Tree (2)



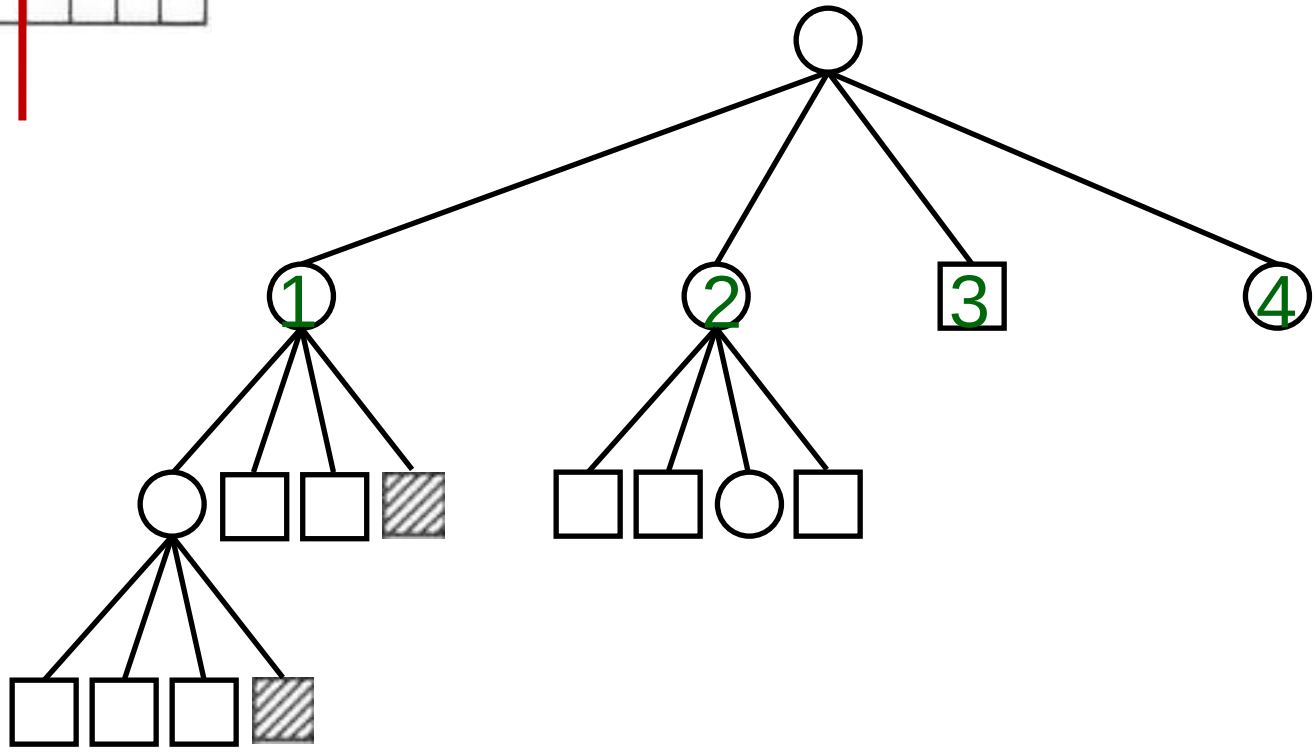
Primitives



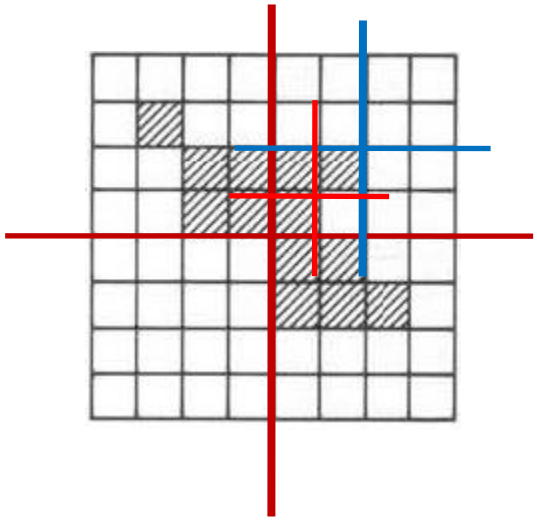
Representation using Tree (2)



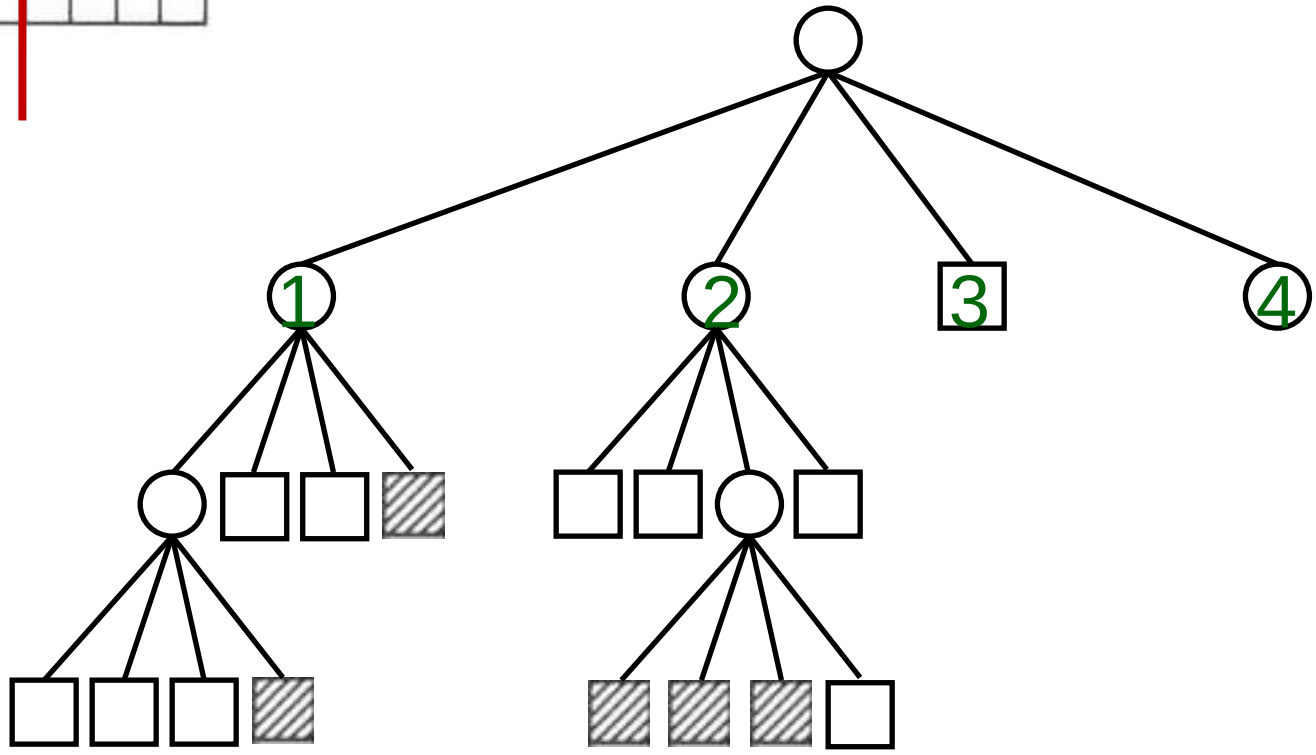
Primitives



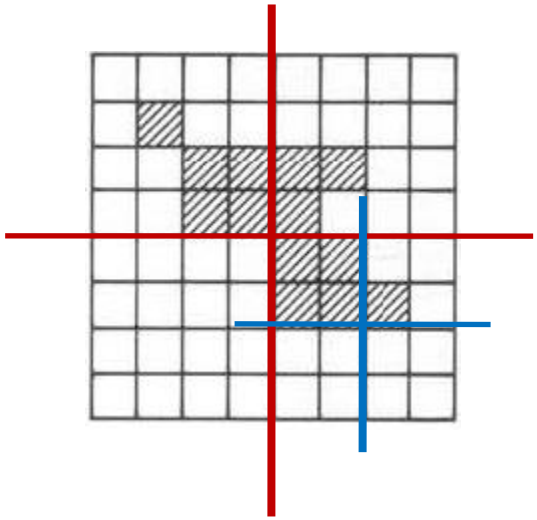
Representation using Tree (2)



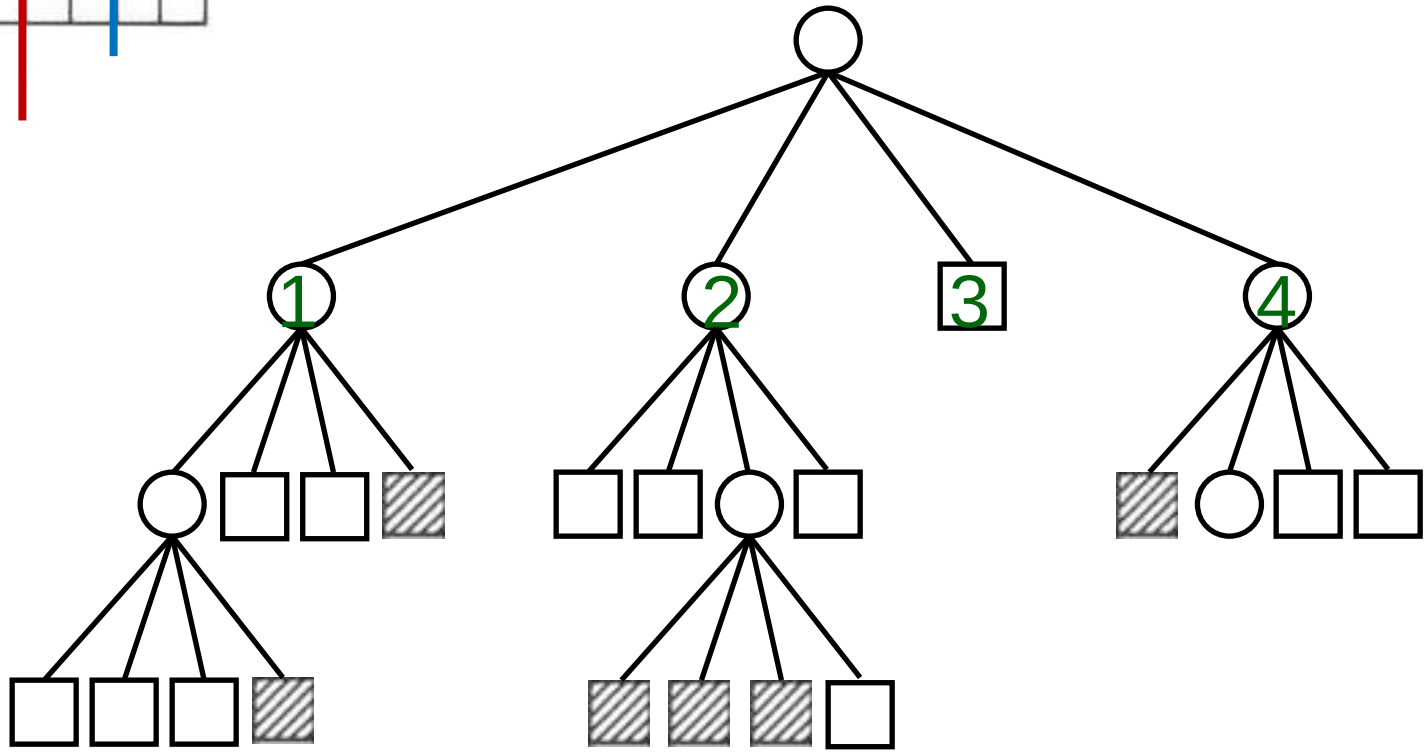
Primitives



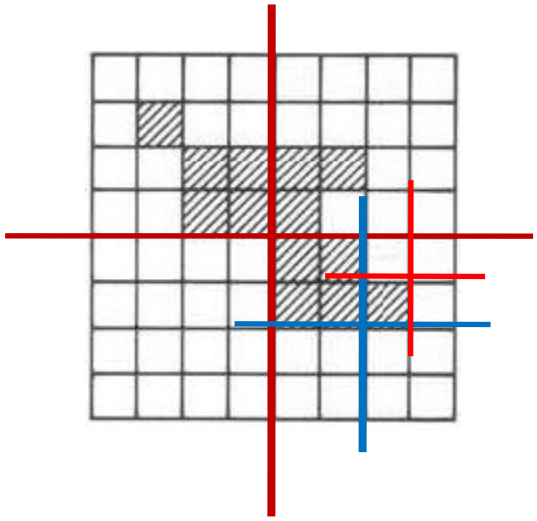
Representation using Tree (2)



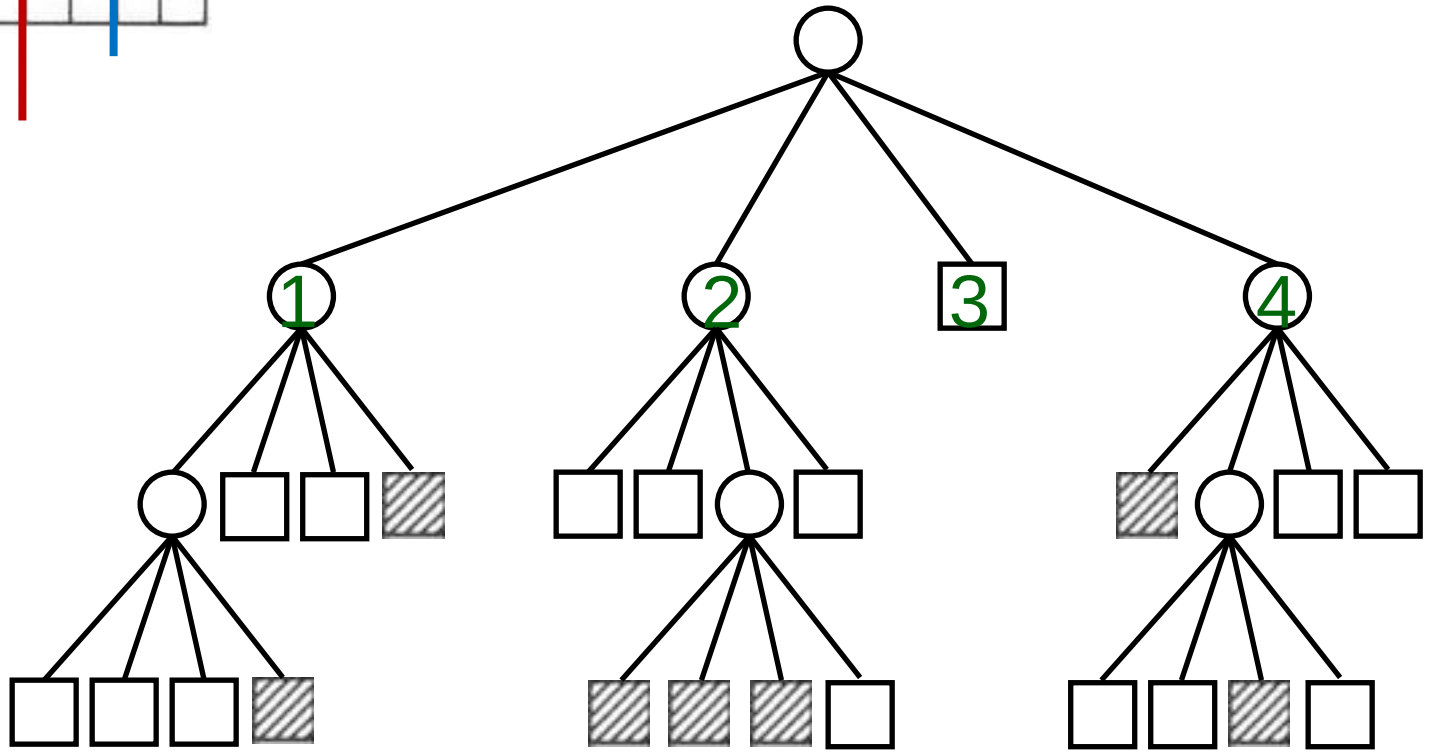
Primitives



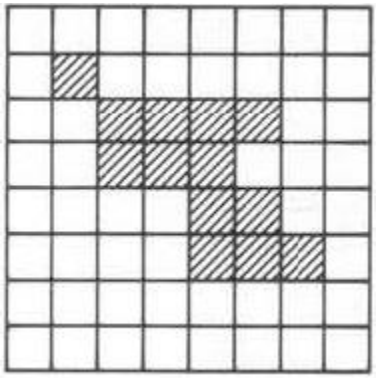
Representation using Tree (2)



Primitives



Representation using Tree (2)



Primitives