# Decidable properties of CFL

Several undecidable are there …

- Membership question.
- Empty?
- Infinite/not ?

# Membership question

- Given the CFG $G$ and string $w$, we ask is $w \in L(G)$?
- There is a $O(n^3)$ algorithm where $|w| = n$, which is called the CYK algorithm.
  - This is a parsing technique whereby one can create the parse tree if the string is in the language.
  - Since this works for any CFG (not restricted to a subclass), this is one of universal parsers.

- If $w = \epsilon$, we verify to find whether $S$ is nullable or not.
- Else, we convert the CFG in to CNF first.
- With CNF form the parse tree is a binary tree.
- And the string $w$ can be derived in exactly $2|w| - 1$ steps.
- The parse tree will have exactly this many variables.

- We can list all possible derivations having $2|w| - 1$ steps.
- We verify whether, any, gave the string.
- But, this is an exponential time algorithm.

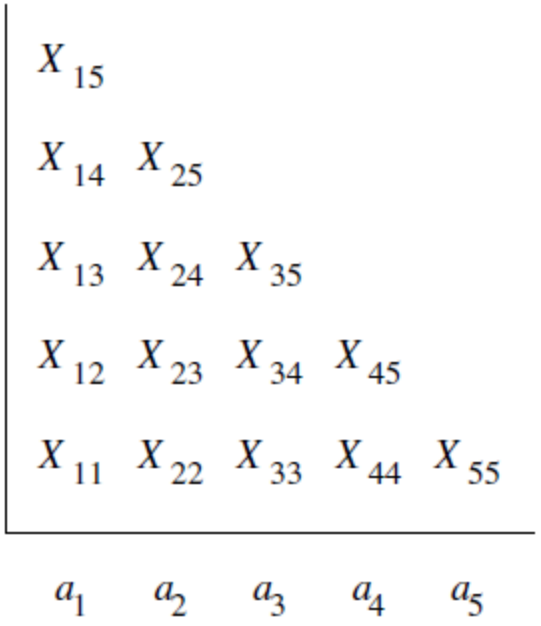
- There is a much more efficient technique based on the idea of “dynamic programming”.
- This is called the CYK algorithm.
- Also called the table-filling or tabulation algorithm.

[3]It is named after three people, each of whom independently discovered essentially the same idea: J. Cocke, D. Younger, and T. Kasami.

# CYK algorithm

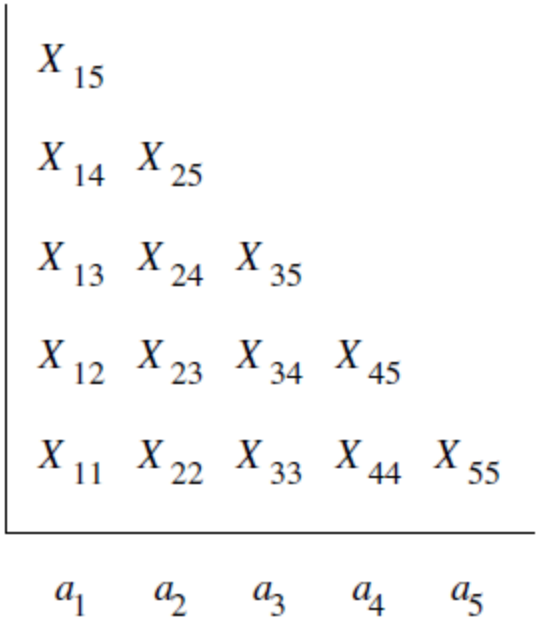
- Let $w = a_1 a_2 \cdots a_n$ be the given string.
- We fill a table, as shown, for example when $w = a_1 a_2 \cdots a_5$

The table entry $X_{ij}$ is the set of variables $A$ such that $A \stackrel{*}{\Rightarrow} a_i a_{i+1} \cdots a_j$.



- If $S \in X_{1n}$ then $S \overset{*}{\Rightarrow} w$
- To find $X_{1n}$ we need to fill the table, in a bottom-up fashion.

The table entry $X_{ij}$ is the set of variables $A$ such that $A \overset{*}{\Rightarrow} a_i a_{i+1} \cdots a_j$.

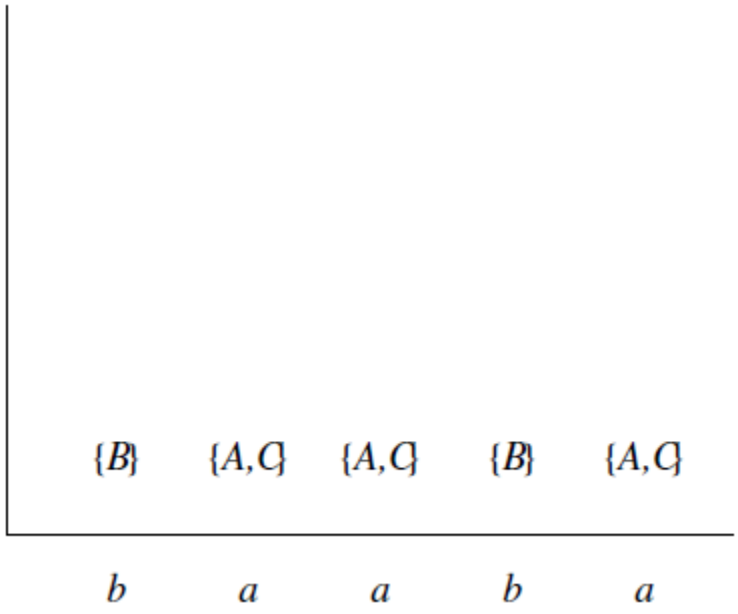


$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string $baaba$

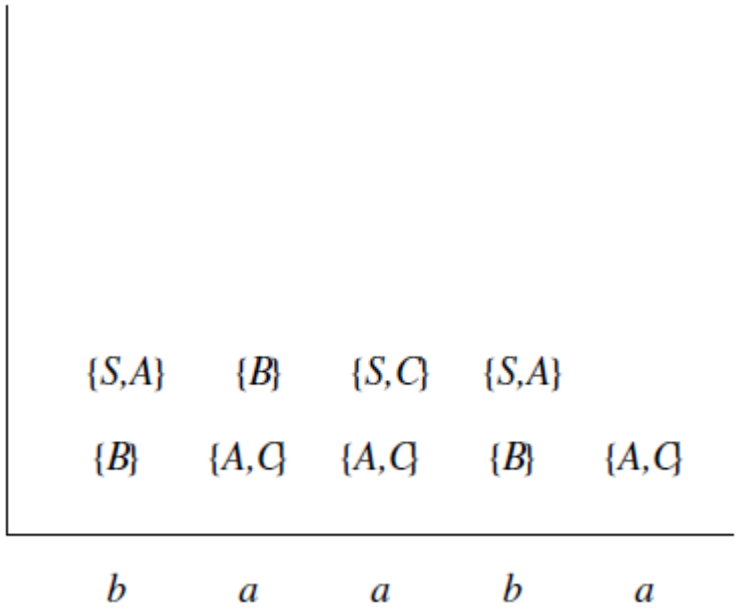
$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string *baaba*



$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string $baaba$



$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string *baaba*

| | | | | |
|---|---|---|---|---|
| – | $\{B\}$ | $\{B\}$ | | |
| $\{S,A\}$ | $\{B\}$ | $\{S,C\}$ | $\{S,A\}$ | |
| $\{B\}$ | $\{A,C\}$ | $\{A,C\}$ | $\{B\}$ | $\{A,C\}$ |
| $b$ | $a$ | $a$ | $b$ | $a$ |

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string *baaba*

| | | | | |
|---|---|---|---|---|
| – | {S,A,C} | | | |
| – | {B} | {B} | | |
| {S,A} | {B} | {S,C} | {S,A} | |
| {B} | {A,C} | {A,C} | {B} | {A,C} |
| *b* | *a* | *a* | *b* | *a* |

$$
\begin{array}{lcl}
S & \rightarrow & AB \mid BC \\
A & \rightarrow & BA \mid a \\
B & \rightarrow & CC \mid b \\
C & \rightarrow & AB \mid a
\end{array}
$$

We shall test for membership in $L(G)$ the string $baaba$

| | | | | |
|---|---|---|---|---|
| $\{S,A,C\}$ | | | | |
| – | $\{S,A,C\}$ | | | |
| – | $\{B\}$ | $\{B\}$ | | |
| $\{S,A\}$ | $\{B\}$ | $\{S,C\}$ | $\{S,A\}$ | |
| $\{B\}$ | $\{A,C\}$ | $\{A,C\}$ | $\{B\}$ | $\{A,C\}$ |
| $b$ | $a$ | $a$ | $b$ | $a$ |

# Parse tree

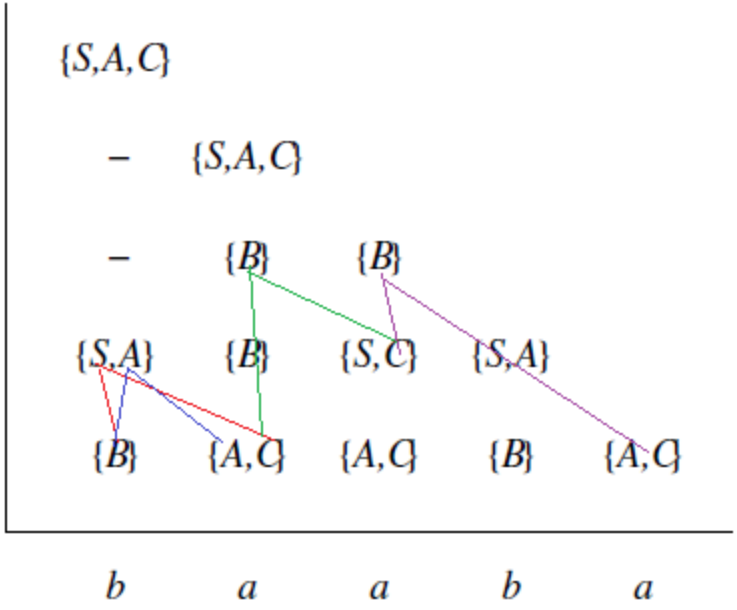
- Parse tree can be found by keeping track of some side information.

We shall test for membership in $L(G)$ the string $baaba$

| | | |
|---|---|---|
| $S$ | $\rightarrow$ | $AB \mid BC$ |
| $A$ | $\rightarrow$ | $BA \mid a$ |
| $B$ | $\rightarrow$ | $CC \mid b$ |
| $C$ | $\rightarrow$ | $AB \mid a$ |

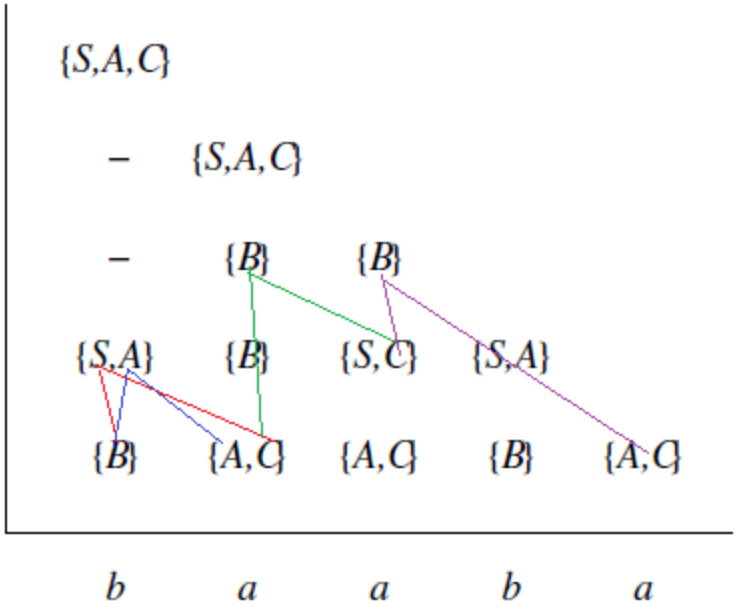
We shall test for membership in $L(G)$ the string $baaba$

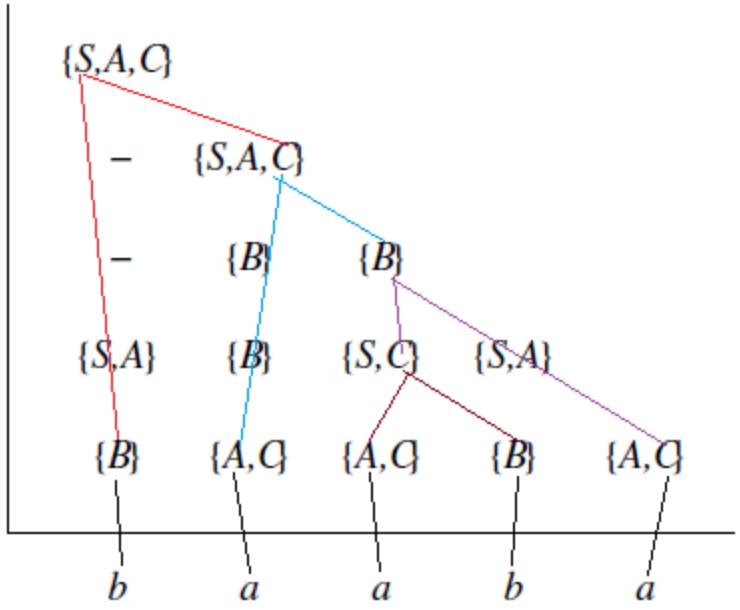
| | | | |
|---|---|---|---|
| $S$ | $\rightarrow$ | $AB$ | $BC$ |
| $A$ | $\rightarrow$ | $BA$ | $a$ |
| $B$ | $\rightarrow$ | $CC$ | $b$ |
| $C$ | $\rightarrow$ | $AB$ | $a$ |



We shall test for membership in $L(G)$ the string *baaba*

| | | |
|---|---|---|
| $S$ | $\rightarrow$ | $AB \mid BC$ |
| $A$ | $\rightarrow$ | $BA \mid a$ |
| $B$ | $\rightarrow$ | $CC \mid b$ |
| $C$ | $\rightarrow$ | $AB \mid a$ |



$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

1. Is $w = aaabb$ in $L(G)$ ?
2. Is $w = aaabbb$ in $L(G)$ ?

The given string is aaabb.

The grammar is CNF is :

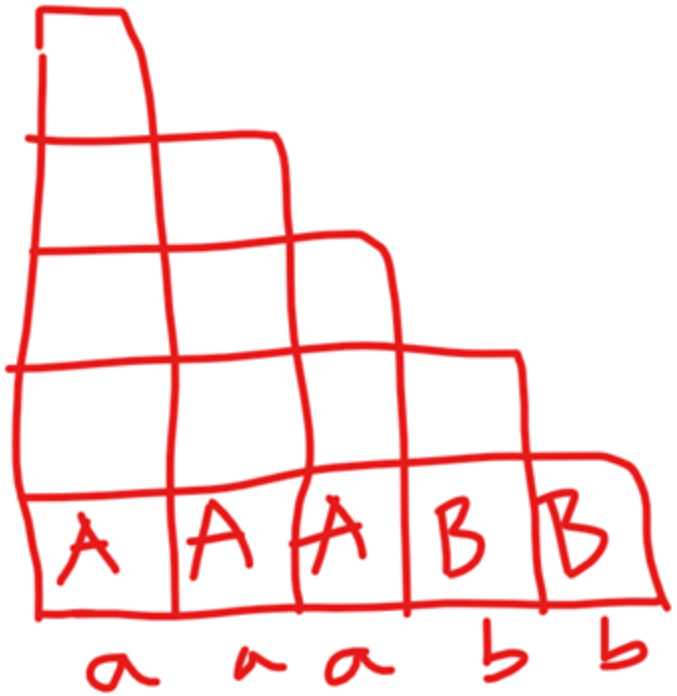
$S \rightarrow AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$




- Complete this..

# Time complexity of CYK

- $O(n^3)$

# Empty ?

- This is easy.
- Is S generating ?

# Infinite or not?

- **Algorithm**.

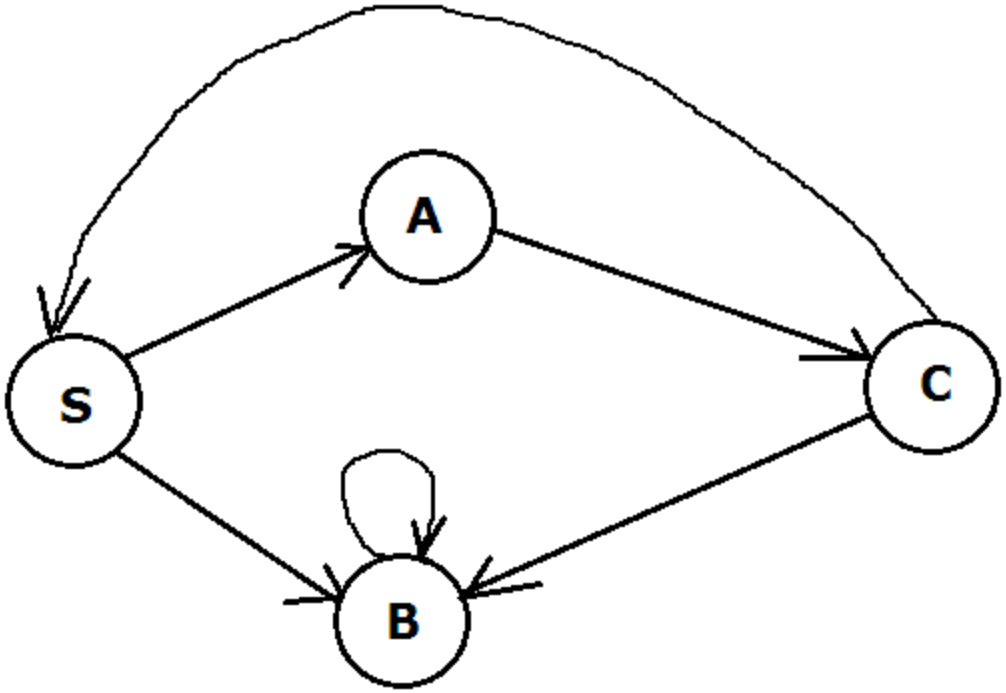
1. Remove useless symbols.
2. Remove unit and $\epsilon$ productions
3. Create dependency graph for variables
4. If there is a loop in the dependency graph then the language is infinite, else not.

# Example

- $S \rightarrow AB, A \rightarrow aCb|a, B \rightarrow bB|b, C \rightarrow cBS$

# Example

- $S \rightarrow AB, A \rightarrow aCb|a, B \rightarrow bB|b, C \rightarrow cBS$



# Some undecidable properties ☹

- Let $G_1$ and $G_2$ be two CFGs.
- Is $L(G_1) = \Sigma^*$ ?
- Is $L(G_1)$ is regular ?
- Is $L(G_1) \subseteq L(G_2)$ ?
- Is $L(G_1) = L(G_2)$ ?
- Is $L(G_1) \cap L(G_2) = \phi$ ?
- Is $L(G_1)$ ambiguous? (inherent ambiguity)
- Is $G_1$ ambiguous?