

Languages and Machines (Module 7 TCS+IAM)  
L&M 5: Chomsky Normal Form and CYK parsing  
Ch 4:1-6

Alexander Skopalik

Discrete Mathematics and Mathematical Programming, Applied Mathematics, University  
of Twente

Lecture 5

# Contents

- 1 Syntax of programming languages
  - Ambiguity
  - Naive Parsing
- 2 Chomsky normal form (CNF)
  - Grammar Transformations
  - CNF definition
  - Transformation to CNF
  - Optimisations
- 3 The CYK parsing-algorithm

# Summary

## Context-free Grammars (CFGs)

- Distinction *variables* ( $V$  with  $S \in V$ ) and *symbols* ( $\Sigma$ )
- Variables are similar to states of an NFA
- Set of *rules* of the form  $A \rightarrow w$ ;  
 $A \in V, w \in (V \cup \Sigma)^*$

# Summary

## Context-free Grammars (CFGs)

- Distinction *variables* ( $V$  with  $S \in V$ ) and *symbols* ( $\Sigma$ )
- Variables are similar to states of an NFA
- Set of *rules* of the form  $A \rightarrow w$ ;  
 $A \in V, w \in (V \cup \Sigma)^*$

## Derivations and derivation trees

- One-to-one-relationship *leftmost* derivations and derivation trees

# Summary

## Context-free Grammars (CFGs)

- Distinction *variables* ( $V$  with  $S \in V$ ) and *symbols* ( $\Sigma$ )
- Variables are similar to states of an NFA
- Set of *rules* of the form  $A \rightarrow w$ ;  
 $A \in V, w \in (V \cup \Sigma)^*$

## Derivations and derivation trees

- One-to-one-relationship *leftmost* derivations and derivation trees

## Language of a CFG

- Set of derivable sentences:  $S \Rightarrow^* w, w \in \Sigma^*$

# Summary

## Context-free Grammars (CFGs)

- Distinction *variables* ( $V$  with  $S \in V$ ) and *symbols* ( $\Sigma$ )
- Variables are similar to states of an NFA
- Set of *rules* of the form  $A \rightarrow w$ ;  
 $A \in V, w \in (V \cup \Sigma)^*$

## Derivations and derivation trees

- One-to-one-relationship *leftmost* derivations and derivation trees

## Language of a CFG

- Set of derivable sentences:  $S \Rightarrow^* w, w \in \Sigma^*$

## Special case: regular grammars

- Translation from and to NFAs
- Regular languages subset of context-free languages

# Summary

## Context-free Grammars (CFGs)

- Distinction *variables* ( $V$  with  $S \in V$ ) and *symbols* ( $\Sigma$ )
- Variables are similar to states of an NFA
- Set of *rules* of the form  $A \rightarrow w$ ;  
 $A \in V, w \in (V \cup \Sigma)^*$

## Derivations and derivation trees

- One-to-one-relationship *leftmost* derivations and derivation trees

## Language of a CFG

- Set of derivable sentences:  $S \Rightarrow^* w, w \in \Sigma^*$

## Special case: regular grammars

- Translation from and to NFAs
- Regular languages subset of context-free languages

## Ambiguity of a grammar

- Sentence(s) with more than one derivation tree

# Contents

- 1 Syntax of programming languages
  - Ambiguity
  - Naive Parsing
- 2 Chomsky normal form (CNF)
  - Grammar Transformations
  - CNF definition
  - Transformation to CNF
  - Optimisations
- 3 The CYK parsing-algorithm

# Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned}
 \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\
 \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\
 \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\
 \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\
 \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid ( \langle \text{Expr} \rangle )
 \end{aligned}$$

# Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned}
 \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\
 \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\
 \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\
 \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\
 \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid ( \langle \text{Expr} \rangle )
 \end{aligned}$$

- *Backus-Naur-Vorm* (BNF): notation-agreements
  - Non-terminals in angular brackets, for example  $\langle \text{Number} \rangle$
  - Subscript  $_{opt}$  (optional) for “λ-choice:”
    - $\langle A \rangle \rightarrow \langle B \rangle_{opt}$  short for  $\langle A \rangle \rightarrow \langle B \rangle \mid \lambda$
    - $\langle A \rangle \rightarrow \langle B \rangle_{opt} \langle C \rangle$  short for  $\langle A \rangle \rightarrow \langle B \rangle \langle C \rangle \mid \langle C \rangle$
  - Many variations and extensions exist

# Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned}
 \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\
 \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\
 \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\
 \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\
 \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid ( \langle \text{Expr} \rangle )
 \end{aligned}$$

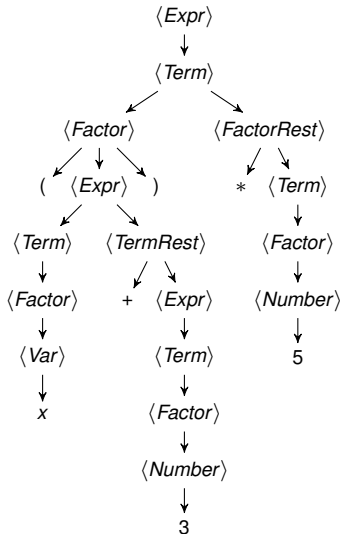
- *Backus-Naur-Vorm* (BNF): notation-agreements
  - Non-terminals in angular brackets, for example  $\langle \text{Number} \rangle$
  - Subscript  $_{opt}$  (optional) for “λ-choice:”
    - $\langle A \rangle \rightarrow \langle B \rangle_{opt}$  short for  $\langle A \rangle \rightarrow \langle B \rangle \mid \lambda$
    - $\langle A \rangle \rightarrow \langle B \rangle_{opt} \langle C \rangle$  short for  $\langle A \rangle \rightarrow \langle B \rangle \langle C \rangle \mid \langle C \rangle$
  - Many variations and extensions exist
- See book (appendix IV) for complete BNF grammar for Java programs

# Syntax of programming languages: example

Parse (i.e., find a derivation for):  $(x + 3) * 5$

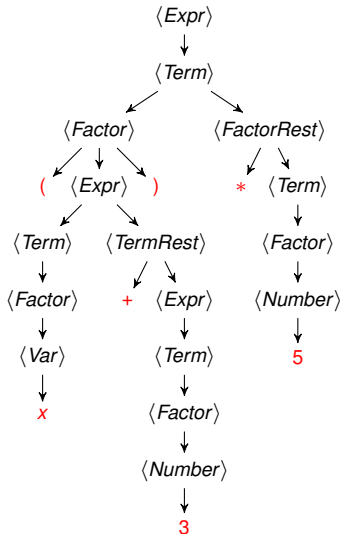
# Syntax of programming languages: example

Parse (i.e., find a derivation for):  $(x + 3) * 5$



# Syntax of programming languages: example

Parse (i.e., find a derivation for):  $(x + 3) * 5$



# Example of ambiguity

- Sentence: *Look at the dog with one eye*

## Example of ambiguity

- Sentence: *Look at the dog with one eye*
- Grammar:

*Sentence* → *VerbPhrase NounPhrase RelPhrase* |  
                   *VerbPhrase NounPhrase AdvPhrase*  
*NounPhrase* → *Pr. Det. PropoerNoun*  
                   ...

# Example of ambiguity

- Sentence: *Look at the dog with one eye*

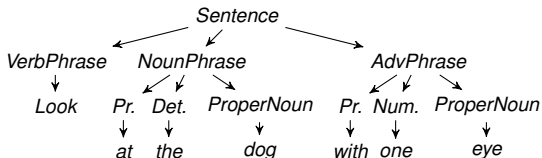
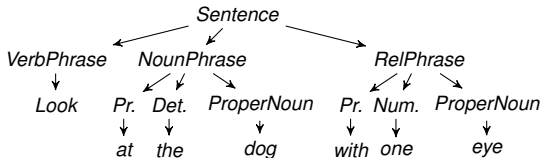
- Grammar:

*Sentence* → *VerbPhrase NounPhrase RelPhrase* |  
*VerbPhrase NounPhrase AdvPhrase*

*NounPhrase* → *Pr. Det. ProperNoun*

...

- Derivation trees:



# Example of ambiguity

- Sentence: *Look at the dog with one eye*

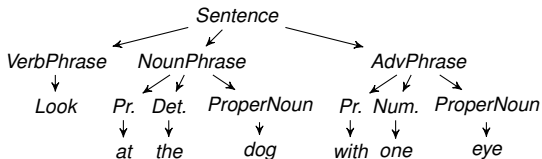
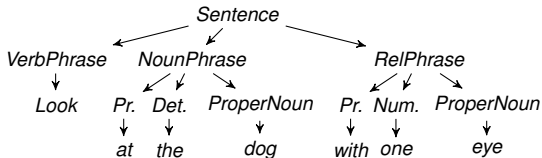
- Grammar:

*Sentence* → *VerbPhrase NounPhrase RelPhrase* |  
*VerbPhrase NounPhrase AdvPhrase*

*NounPhrase* → *Pr. Det. ProperNoun*

...

- Derivation trees:



# Definition of (un)ambiguity

## Definition

A CFG is *ambiguous* if a word exists in  $L(G)$  with two distinct derivation trees.

# Definition of (un)ambiguity

## Definition

A CFG is *ambiguous* if a word exists in  $L(G)$  with two distinct derivation trees.

## Remarks

- Ambiguity results from the grammar, not from the language  
The following language has an ambiguous and an unambiguous grammar:

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$L(G_1) = L(G_2) = a^+$  holds; but  $G_1$  is ambiguous,  $G_2$  is not

# Definition of (un)ambiguity

## Definition

A CFG is *ambiguous* if a word exists in  $L(G)$  with two distinct derivation trees.

## Remarks

- Ambiguity results from the grammar, not from the language  
The following language has an ambiguous and an unambiguous grammar:

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$L(G_1) = L(G_2) = a^+$  holds; but  $G_1$  is ambiguous,  $G_2$  is not

- **Fact:** There are also languages with *only* ambiguous grammars  
Example:  $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$

# Definition of (un)ambiguity

## Definition

A CFG is *ambiguous* if a word exists in  $L(G)$  with two distinct derivation trees.

## Remarks

- Ambiguity results from the grammar, not from the language  
The following language has an ambiguous and an unambiguous grammar:  

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$$L(G_1) = L(G_2) = a^+ \text{ holds; but } G_1 \text{ is ambiguous, } G_2 \text{ is not}$$
- **Fact:** There are also languages with *only* ambiguous grammars  
Example:  $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$
- Unambiguous grammars exist for regular languages
- Programming languages have unambiguous grammars

# The principle of parsing

## Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

# The principle of parsing

## Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

## Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only  $S$ , we are done
- 2 Search right side of a rule that matches part of the sentence

Success?

Failure?

# The principle of parsing

## Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

## Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only  $S$ , we are done
- 2 Search right side of a rule that matches part of the sentence

### Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

### Failure?

# The principle of parsing

## Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

## Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only  $S$ , we are done
- 2 Search right side of a rule that matches part of the sentence

### Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

### Failure?

- Go back to the previous phase
- Try the next possibility

# The principle of parsing

## Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

## Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only  $S$ , we are done
- 2 Search right side of a rule that matches part of the sentence

### Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

### Failure?

- Go back to the previous phase
- Try the next possibility

(backtracking)

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:

## Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$

## Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab$

## Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab$

## Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab$

## Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \quad \zeta$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \quad \text{⚡}$   
 $\leftarrow AaAb$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\qquad \qquad \qquad \qquad \qquad \qquad \leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\qquad \qquad \qquad \leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \end{matrix}$



# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$







# Example of naive parsing

• Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$

• Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAb$



# Example of naive parsing

• Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$

• Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$



# Example of naive parsing

• Grammar:  $S \rightarrow aSb \mid aA$      $A \rightarrow aA \mid a$

• Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow Sb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aSb$

# Example of naive parsing

• Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$

• Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$

$\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow Sb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

$\leftarrow aSb \leftarrow S \quad \checkmark$

# Example of naive parsing

- Grammar:  $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language:  $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing:  $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow Sb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$   
 $\leftarrow aSb \leftarrow S \quad \checkmark$

- The complexity of this algorithm is not acceptable! Why?
  - It becomes even worse if there is also  $\lambda$  on the right side

# Contents

- 1 Syntax of programming languages
  - Ambiguity
  - Naive Parsing
- 2 Chomsky normal form (CNF)
  - Grammar Transformations
  - CNF definition
  - Transformation to CNF
  - Optimisations
- 3 The CYK parsing-algorithm

# Normal forms of grammars

**Known:** A language can have several grammars

- Some are more suitable for parsing than others

**Normal form:** restriction on the right side of the rules

- Example: never  $aAb$  but always  $AB$ ; never  $\lambda$
- Simpler to analyze
- More efficient parsing algorithms

# Normal forms of grammars

**Known:** A language can have several grammars

- Some are more suitable for parsing than others

**Normal form:** restriction on the right side of the rules

- Example: never  $aAb$  but always  $AB$ ; never  $\lambda$
- Simpler to analyze
- More efficient parsing algorithms

**Automatic transformation to normal form:**

- Through *step-by-step transformation* of the grammar

# Examples of these grammar transformations

Take  $G$ :  $S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$

- Adding a new start symbol  $S'$  results in:

# Examples of these grammar transformations

Take  $G$ :  $S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$

- Adding a new start symbol  $S'$  results in:

$$G_1: \quad S' \rightarrow S \quad S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$$

(rationale: the new start symbol will not occur recursively)

# Examples of these grammar transformations

Take  $G$ :  $S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$

- Adding a new start symbol  $S'$  results in:

$$G_1: S' \rightarrow S \quad S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$$

(rationale: the new start symbol will not occur recursively)

- Inlining the  $S$ -rule in  $A$  results in:

# Examples of these grammar transformations

Take  $G$ :  $S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$

- Adding a new start symbol  $S'$  results in:

$$G_1: \quad S' \rightarrow S \quad S \rightarrow aS \mid Ab \quad A \rightarrow cSb \mid \lambda$$

(rationale: the new start symbol will not occur recursively)

- Inlining the  $S$ -rule in  $A$  results in:

$$G_3: \quad S \rightarrow aS \mid Ab \quad A \rightarrow caSb \mid cAbb \mid \lambda$$

# Simple transformations of grammars

Let  $G = \langle V, \Sigma, P, S \rangle$  be a given grammar.

## Lemma: Add start symbol

Take  $S' \notin V$ , a new symbol. Let

$G_1 := \langle V \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S' \rangle$ .

Then  $\mathcal{L}(G_1) = \mathcal{L}(G)$ .

# Simple transformations of grammars

Let  $G = \langle V, \Sigma, P, S \rangle$  be a given grammar.

## Lemma: Add start symbol

Take  $S' \notin V$ , a new symbol. Let

$G_1 := \langle V \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S' \rangle$ .

Then  $\mathcal{L}(G_1) = \mathcal{L}(G)$ .

## Lemma: “Inlining” rules

Suppose that  $A \rightarrow u B v$  and  $B \rightarrow w_1 \mid w_2 \mid \dots \mid w_n$  in  $P$ .

Let  $G_3 := \langle V, \Sigma, P_3, S \rangle$  with

$$P_3 = (P \setminus \{A \rightarrow u B v\}) \cup \{A \rightarrow u w_1 v \mid u w_2 v \mid \dots \mid u w_n v\} .$$

Then  $\mathcal{L}(G_3) = \mathcal{L}(G)$ .

# Chomsky normal form (CNF)

## Definition

$G = \langle V, \Sigma, P, S \rangle$  is in *Chomsky normal form* if all rules are of one of the following forms:

- $A \rightarrow BC$ , ór
- $A \rightarrow a$ , ór
- $S \rightarrow \lambda$

met  $B, C \in V \setminus \{S\}$  (so  $S$  is not recursive)

# Chomsky normal form (CNF)

## Definition

$G = \langle V, \Sigma, P, S \rangle$  is in *Chomsky normal form* if all rules are of one of the following forms:

- $A \rightarrow BC$ , ór
- $A \rightarrow a$ , ór
- $S \rightarrow \lambda$

met  $B, C \in V \setminus \{S\}$  (so  $S$  is not recursive)

For example for:  $S \rightarrow aSb \mid aA$   
 $A \rightarrow aA \mid a$

this is a Chomsky normal form:

# Chomsky normal form (CNF)

## Definition

$G = \langle V, \Sigma, P, S \rangle$  is in *Chomsky normal form* if all rules are of one of the following forms:

- $A \rightarrow BC$ , ór
- $A \rightarrow a$ , ór
- $S \rightarrow \lambda$

met  $B, C \in V \setminus \{S\}$  (so  $S$  is not recursive)

For example for:  $S \rightarrow aSb \mid aA$

$A \rightarrow aA \mid a$

this is a Chomsky normal form:  $S \rightarrow S_1 S_2 \mid S_1 A$

$S_1 \rightarrow a$

$S_2 \rightarrow S_3 S_4$

# Chomsky normal form (CNF)

## Definition

$G = \langle V, \Sigma, P, S \rangle$  is in *Chomsky normal form* if all rules are of one of the following forms:

- $A \rightarrow BC$ , or
- $A \rightarrow a$ , or
- $S \rightarrow \lambda$

met  $B, C \in V \setminus \{S\}$  (so  $S$  is not recursive)

For example for:  $S \rightarrow aSb \mid aA$   
 $A \rightarrow aA \mid a$

this is a Chomsky normal form:  $S \rightarrow S_1 S_2 \mid S_1 A$   
 $S_1 \rightarrow a$   
 $S_2 \rightarrow S_3 S_4$   
 $S_3 \rightarrow S_1 S_2 \mid S_1 A$   
 $S_4 \rightarrow b$   
 $A \rightarrow S_1 A \mid a$

# Transformation to CNF

- **Advantages Chomsky normal form:**
  - Derivations are non-contracting (words do not get shorter)
  - Derivation of  $w$  at most  $2n - 1$  steps ( $n = |w|$ )
  - Derivation tree is binary, depth at most  $n$
  - Efficient parsing algorithm

# Transformation to CNF

- **Advantages Chomsky normal form:**
  - Derivations are non-contracting (words do not get shorter)
  - Derivation of  $w$  at most  $2n - 1$  steps ( $n = |w|$ )
  - Derivation tree is binary, depth at most  $n$
  - Efficient parsing algorithm
- **Necessary steps to achieve CNF:**
  - 1 Eliminate recursion in start symbol
  - 2 Eliminate  $\lambda$ -rules (except for  $S$ )
  - 3 Eliminate “chain-rules”  $A \rightarrow B$
  - 4 Finalize rules  $A \rightarrow w$  with  $|w| \geq 2$  and  $w \neq BC$

# Transformation to CNF

- **Advantages Chomsky normal form:**
  - Derivations are non-contracting (words do not get shorter)
  - Derivation of  $w$  at most  $2n - 1$  steps ( $n = |w|$ )
  - Derivation tree is binary, depth at most  $n$
  - Efficient parsing algorithm
- **Necessary steps to achieve CNF:**
  - 1 Eliminate recursion in start symbol
  - 2 Eliminate  $\lambda$ -rules (except for  $S$ )
  - 3 Eliminate “chain-rules”  $A \rightarrow B$
  - 4 Finalize rules  $A \rightarrow w$  with  $|w| \geq 2$  and  $w \neq BC$
- **Clean up:**
  - 5 Eliminate non-terminating variables
  - 6 Eliminate unreachable variables

# Non-contracting grammar: eliminate null-rules

- **Example**

- Grammar with null-rule:

$$G_1 : \quad S \rightarrow SaB \mid aB \quad B \rightarrow bB \mid \lambda$$

- Equivalent grammar without null-rule (non-contracting):

# Non-contracting grammar: eliminate null-rules

- **Example**

- Grammar with null-rule:

$$G_1 : \quad S \rightarrow SaB \mid aB \quad B \rightarrow bB \mid \lambda$$

- Equivalent grammar without null-rule (non-contracting):

$$G_2 : \quad S \rightarrow SaB \mid Sa \mid aB \mid a \quad B \rightarrow bB \mid b$$

# Non-contracting grammar: eliminate null-rules

## • Example

- Grammar with null-rule:

$$G_1 : \quad S \rightarrow SaB \mid aB \quad B \rightarrow bB \mid \lambda$$

- Equivalent grammar without null-rule (non-contracting):

$$G_2 : \quad S \rightarrow SaB \mid Sa \mid aB \mid a \quad B \rightarrow bB \mid b$$

## • Transformation-idea

- 1 Assume that the start symbol is non-recursive
- 2 Collect all variables  $L$  with a derivation to  $\lambda$ ;
- 3 For all rules  $A \rightarrow w$ :
  - If  $w$  is equal to  $w_1 A_1 w_2 A_2 w_3 \cdots w_k A_k w_{k+1}$  (with  $A_i \in L$ )
  - Add a rule  $A \rightarrow w_1 w_2 \cdots w_k w_{k+1}$
- 4 Remove all null-rules except  $S \rightarrow \lambda$

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$

$Prev := \emptyset$

**while** ( $Prev \neq Null$ ):

$Prev := Null$

$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$$

$$Prev := \emptyset$$

**while** ( $Prev \neq Null$ ):

$$Prev := Null$$

$$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$$

- Example grammar:

$$S \rightarrow A C A$$

$$B \rightarrow b B \mid b$$

$$A \rightarrow a A a \mid B \mid C$$

$$C \rightarrow c C \mid \lambda$$

- Iterations algorithm:

<i>Round</i>	<i>Null</i>	<i>Prev</i>
0		

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$$

$$Prev := \emptyset$$

**while** ( $Prev \neq Null$ ):

$$Prev := Null$$

$$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$$

- Example grammar:

$$S \rightarrow A C A$$

$$B \rightarrow b B \mid b$$

$$A \rightarrow a A a \mid B \mid C$$

$$C \rightarrow c C \mid \lambda$$

- Iterations algorithm:

<i>Round</i>	<i>Null</i>	<i>Prev</i>
0	{C}	$\emptyset$
1		

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$$

$$Prev := \emptyset$$

**while** ( $Prev \neq Null$ ):

$$Prev := Null$$

$$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$$

- Example grammar:

$$S \rightarrow A C A$$

$$B \rightarrow b B \mid b$$

$$A \rightarrow a A a \mid B \mid C$$

$$C \rightarrow c C \mid \lambda$$

- Iterations algorithm:

<i>Round</i>	<i>Null</i>	<i>Prev</i>
0	{C}	$\emptyset$
1	{A, C}	{C}
2		

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$$

$$Prev := \emptyset$$

**while** ( $Prev \neq Null$ ):

$$Prev := Null$$

$$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$$

- Example grammar:

$$S \rightarrow A C A$$

$$B \rightarrow b B \mid b$$

$$A \rightarrow a A a \mid B \mid C$$

$$C \rightarrow c C \mid \lambda$$

- Iterations algorithm:

<i>Round</i>	<i>Null</i>	<i>Prev</i>
0	{C}	$\emptyset$
1	{A, C}	{C}
2	{S, A, C}	{A, C}
3		

# Step 1: Collect variables with $\lambda$ -derivation

Algorithm: backwards starting from null-rules

$$Null := \{A \in V \mid (A \rightarrow \lambda) \in P\}$$

$$Prev := \emptyset$$

**while** ( $Prev \neq Null$ ):

$$Prev := Null$$

$$Null := Null \cup \{A \in V \mid \exists w \in Null^* : (A \rightarrow w) \in P\}$$

- Example grammar:

$$S \rightarrow A C A$$

$$B \rightarrow b B \mid b$$

$$A \rightarrow a A a \mid B \mid C$$

$$C \rightarrow c C \mid \lambda$$

- Iterations algorithm:

<i>Round</i>	<i>Null</i>	<i>Prev</i>
0	{C}	$\emptyset$
1	{A, C}	{C}
2	{S, A, C}	{A, C}
3	{S, A, C}	{S, A, C}

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

$$S \rightarrow \lambda \mid A \mid C \mid A C \mid C A \mid A A \mid A C A$$

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

$$S \rightarrow \lambda \mid A \mid C \mid A C \mid C A \mid A A \mid A C A$$

$$A \rightarrow a a \mid a A a \mid B \mid \lambda \mid C$$

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

$$S \rightarrow \lambda \mid A \mid C \mid A C \mid C A \mid A A \mid A C A$$

$$A \rightarrow a a \mid a A a \mid B \mid \lambda \mid C$$

$$B \rightarrow b B \mid b$$

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

$$\begin{aligned}
 S &\rightarrow \lambda \mid A \mid C \mid A C \mid C A \mid A A \mid A C A \\
 A &\rightarrow a a \mid a A a \mid B \mid \lambda \mid C \\
 B &\rightarrow b B \mid b \\
 C &\rightarrow c \mid c C \mid \lambda
 \end{aligned}$$

- Step 3: Remove rules  $A \rightarrow \lambda$  (except  $S \rightarrow \lambda$ )

## Step 2 and 3: Eliminate variables with $\lambda$ -derivation

- Example grammar:  $S \rightarrow A C A$   
 $A \rightarrow a A a \mid B \mid C$   
 $B \rightarrow b B \mid b$   
 $C \rightarrow c C \mid \lambda$
- Step 2: Remove **all** combinations of variables  $\{S, A, C\}$ :

$$\begin{aligned}
 S &\rightarrow \lambda \mid A \mid C \mid A C \mid C A \mid A A \mid A C A \\
 A &\rightarrow a a \mid a A a \mid B \mid \lambda \mid C \\
 B &\rightarrow b B \mid b \\
 C &\rightarrow c \mid c C \mid \lambda
 \end{aligned}$$

- Step 3: Remove rules  $A \rightarrow \lambda$  (except  $S \rightarrow \lambda$ )

# Eliminate chain-rules

- Example

- Grammar with chain-rules:

$$G_1 : S \rightarrow A \mid aB \quad A \rightarrow Ab \mid B \quad B \rightarrow AA \mid b$$

- Equivalent grammar without chain-rule:

# Eliminate chain-rules

- Example

- Grammar with chain-rules:

$$G_1 : S \rightarrow A \mid aB \quad A \rightarrow Ab \mid B \quad B \rightarrow AA \mid b$$

- Equivalent grammar without chain-rule:

$$G_2 : S \rightarrow Ab \mid AA \mid b \mid aB \quad A \rightarrow Ab \mid AA \mid b \quad B \rightarrow AA \mid b$$

# Eliminate chain-rules

- Example

- Grammar with chain-rules:

$$G_1 : S \rightarrow A \mid aB \quad A \rightarrow Ab \mid B \quad B \rightarrow AA \mid b$$

- Equivalent grammar without chain-rule:

$$G_2 : S \rightarrow Ab \mid AA \mid b \mid aB \quad A \rightarrow Ab \mid AA \mid b \quad B \rightarrow AA \mid b$$

- Transformation-idea

- 1 Assume that the grammar is non-contracting
- 2 Collect the reachable variables  $Chain(A) = \{B \in V \mid A \xRightarrow{*} B\}$
- 3 Create rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

# Step 1: compute chain-variables

Algorithm: forwards from chain-rules

$Chain(A) := \{A\}$

$Prev := \emptyset$

**while** ( $Chain(A) \neq Prev$ ):

$New := Chain(A) \setminus Prev$

$Prev := Chain(A)$

$Chain(A) := Chain(A) \cup \{C \in V \mid \exists B \in New : (B \rightarrow C) \in P\}$

# Step 1: compute chain-variables

Algorithm: forwards from chain-rules

$Chain(A) := \{A\}$

$Prev := \emptyset$

**while** ( $Chain(A) \neq Prev$ ):

$New := Chain(A) \setminus Prev$

$Prev := Chain(A)$

$Chain(A) := Chain(A) \cup \{C \in V \mid \exists B \in New : (B \rightarrow C) \in P\}$

- Example grammar:

$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$

$B \rightarrow bB \mid b$

$A \rightarrow aa \mid aAa \mid B \mid C$

$C \rightarrow c \mid cC$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$Chain(A) := \{A\}$

$Prev := \emptyset$

**while** ( $Chain(A) \neq Prev$ ):

$New := Chain(A) \setminus Prev$

$Prev := Chain(A)$

$Chain(A) := Chain(A) \cup \{C \in V \mid \exists B \in New : (B \rightarrow C) \in P\}$

- Example grammar:

$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$

$B \rightarrow bB \mid b$

$A \rightarrow aa \mid aAa \mid B \mid C$

$C \rightarrow c \mid cC$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$Chain(A) := \{A\}$

$Prev := \emptyset$

**while** ( $Chain(A) \neq Prev$ ):

$New := Chain(A) \setminus Prev$

$Prev := Chain(A)$

$Chain(A) := Chain(A) \cup \{C \in V \mid \exists B \in New : (B \rightarrow C) \in P\}$

- Example grammar:

$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$

$B \rightarrow bB \mid b$

$A \rightarrow aa \mid aAa \mid B \mid C$

$C \rightarrow c \mid cC$

- Result of algorithm:  $Chain(S) =$   
 $Chain(A) =$   
 $Chain(B) =$   
 $Chain(C) =$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$Chain(A) := \{A\}$

$Prev := \emptyset$

**while** ( $Chain(A) \neq Prev$ ):

$New := Chain(A) \setminus Prev$

$Prev := Chain(A)$

$Chain(A) := Chain(A) \cup \{C \in V \mid \exists B \in New : (B \rightarrow C) \in P\}$

- Example grammar:

$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$        $B \rightarrow bB \mid b$

$A \rightarrow aa \mid aAa \mid B \mid C$        $C \rightarrow c \mid cC$

- Result of algorithm:  $Chain(S) = \{S, A, C, B\}$

$Chain(A) =$

$Chain(B) =$

$Chain(C) =$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$$\text{Chain}(A) := \{A\}$$

$$\text{Prev} := \emptyset$$

**while** ( $\text{Chain}(A) \neq \text{Prev}$ ):

$$\text{New} := \text{Chain}(A) \setminus \text{Prev}$$

$$\text{Prev} := \text{Chain}(A)$$

$$\text{Chain}(A) := \text{Chain}(A) \cup \{C \in V \mid \exists B \in \text{New} : (B \rightarrow C) \in P\}$$

- Example grammar:

$$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA \qquad B \rightarrow bB \mid b$$

$$A \rightarrow aa \mid aAa \mid B \mid C \qquad C \rightarrow c \mid cC$$

- Result of algorithm:
 
$$\begin{aligned} \text{Chain}(S) &= \{S, A, C, B\} \\ \text{Chain}(A) &= \{A, B, C\} \\ \text{Chain}(B) &= \\ \text{Chain}(C) &= \end{aligned}$$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$$\text{Chain}(A) := \{A\}$$

$$\text{Prev} := \emptyset$$

**while** ( $\text{Chain}(A) \neq \text{Prev}$ ):

$$\text{New} := \text{Chain}(A) \setminus \text{Prev}$$

$$\text{Prev} := \text{Chain}(A)$$

$$\text{Chain}(A) := \text{Chain}(A) \cup \{C \in V \mid \exists B \in \text{New} : (B \rightarrow C) \in P\}$$

- Example grammar:

$$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA \qquad B \rightarrow bB \mid b$$

$$A \rightarrow aa \mid aAa \mid B \mid C \qquad C \rightarrow c \mid cC$$

- Result of algorithm:
 
$$\begin{aligned} \text{Chain}(S) &= \{S, A, C, B\} \\ \text{Chain}(A) &= \{A, B, C\} \\ \text{Chain}(B) &= \{B\} \\ \text{Chain}(C) &= \end{aligned}$$

# Step 1: compute chain-variables

## Algorithm: forwards from chain-rules

$$\text{Chain}(A) := \{A\}$$

$$\text{Prev} := \emptyset$$

**while** ( $\text{Chain}(A) \neq \text{Prev}$ ):

$$\text{New} := \text{Chain}(A) \setminus \text{Prev}$$

$$\text{Prev} := \text{Chain}(A)$$

$$\text{Chain}(A) := \text{Chain}(A) \cup \{C \in V \mid \exists B \in \text{New} : (B \rightarrow C) \in P\}$$

- Example grammar:

$$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA \qquad B \rightarrow bB \mid b$$

$$A \rightarrow aa \mid aAa \mid B \mid C \qquad C \rightarrow c \mid cC$$

- Result of algorithm:
 
$$\begin{aligned} \text{Chain}(S) &= \{S, A, C, B\} \\ \text{Chain}(A) &= \{A, B, C\} \\ \text{Chain}(B) &= \{B\} \\ \text{Chain}(C) &= \{C\} \end{aligned}$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$S \rightarrow$

$A \rightarrow$

$B \rightarrow$

$C \rightarrow$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA$

$A \rightarrow$

$B \rightarrow$

$C \rightarrow$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa$$

$$A \rightarrow$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b$$

$$A \rightarrow$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow aa \mid aAa$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow aa \mid aAa \mid bB \mid b$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$B \rightarrow$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$B \rightarrow bB \mid b$$

$$C \rightarrow$$

## Step 2: Create rules without chain variables

- Grammar:

	<i>Chain</i>
$S \rightarrow \lambda \mid A \mid C \mid AC \mid CA \mid AA \mid ACA$	$\{S, A, B, C\}$
$A \rightarrow aa \mid aAa \mid B \mid C$	$\{A, B, C\}$
$B \rightarrow bB \mid b$	$\{B\}$
$C \rightarrow c \mid cC$	$\{C\}$

- Step 2: Rule  $A \rightarrow w$  for all  $B \in Chain(A)$  and  $B \rightarrow w$  ( $w \notin V$ )

$$S \rightarrow \lambda \mid AC \mid CA \mid AA \mid ACA \mid aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$A \rightarrow aa \mid aAa \mid bB \mid b \mid c \mid cC$$

$$B \rightarrow bB \mid b$$

$$C \rightarrow c \mid cC$$

# Finalizing right sides

- **Example**

- Unfinalized grammar:  $S \rightarrow aAb$   
 $A \rightarrow aAb \mid ab$
- Chomsky normal form:

# Finalizing right sides

- **Example**

- Unfinalized grammar:  $S \rightarrow aAb$   
 $A \rightarrow aAb \mid ab$
- Chomsky normal form:  $S \rightarrow D_1 C_1$   
 $A \rightarrow D_1 C_1 \mid D_1 D_2$   
 $C_1 \rightarrow AD_2$   
 $D_1 \rightarrow a$   
 $D_2 \rightarrow b$

# Finalizing right sides

## ● Example

- Unfinalized grammar:  $S \rightarrow aAb$   
 $A \rightarrow aAb \mid ab$
- Chomsky normal form:  $S \rightarrow D_1 C_1$   
 $A \rightarrow D_1 C_1 \mid D_1 D_2$   
 $C_1 \rightarrow A D_2$   
 $D_1 \rightarrow a$   
 $D_2 \rightarrow b$

- **Transformation-idea:** Replace each rule  $A \rightarrow w$  with  $|w| \geq 2$ :
  - Suppose that  $w = a_1 a_2 \cdots a_n$  with  $a_i \in V \cup \Sigma$  for all  $i \in [1, n]$ , and
    - For  $a_i \in V$ : name  $D_i = a_i$ ;
    - For  $a_i \in \Sigma$ : choose a new  $D_i$  and define  $D_i \rightarrow a_i$
  - Name  $C_1 = A$  and choose new variables  $C_2, \dots, C_{n-1}$ ;
  - Define  $C_i \rightarrow D_i C_{i+1}$  for  $i \in [1, n-2]$ , and  $C_{n-1} \rightarrow D_{n-1} D_n$

# Construction of Chomsky normal form

## Theorem

For every grammar  $G$  a grammar  $G'$  exists in Chomsky normal form such that  $L(G') = L(G)$

## Proof

Construct  $G'$  in steps:

- 1 Transform  $G$  into  $G_1$  by adding a new start symbol
- 2 Transform  $G_1$  into  $G_2$  by eliminating null-rules
- 3 Transform  $G_2$  into  $G_3$  by eliminating chain-rules
- 4 Transform  $G_3$  into  $G'$  by finalizing right sides

Every transformation results in an equivalent grammar.

# Useless variables

- Which variables can *not* be used in a derivation?

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bD \mid b$$

# Useless variables

- Which variables can *not* be used in a derivation?

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bD \mid b$$

- $C$  and  $D$  do not *terminate*;

# Useless variables

- Which variables can *not* be used in a derivation?

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bD \mid b$$

- $C$  and  $D$  do not *terminate*; so  $S \rightarrow BS \mid B$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow b$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow b$$

# Useless variables

- Which variables can *not* be used in a derivation?

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bD \mid b$$

- $C$  and  $D$  do not *terminate*; so  $S \rightarrow BS \mid B$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow b$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow b$$

- $A$ ,  $E$  and  $F$  not *reachable*;

# Useless variables

- Which variables can *not* be used in a derivation?

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bD \mid b$$

- $C$  and  $D$  do not *terminate*; so  $S \rightarrow BS \mid B$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow b$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow b$$

- $A$ ,  $E$  and  $F$  not *reachable*; so  $S \rightarrow BS \mid B$

$$B \rightarrow b$$

# Detection of useless variables

Algorithm: detection of termination (backwards)

$term := \{A \in V \mid \exists w \in \Sigma^* : (A \rightarrow w) \in P\}$

$prev := \emptyset$

**while** ( $term \neq prev$ ):

$prev := term$

$term := term \cup \{A \in V \mid \exists w \in (term \cup \Sigma)^* : (A \rightarrow w) \in P\}$

# Detection of useless variables

## Algorithm: detection of termination (backwards)

$$term := \{A \in V \mid \exists w \in \Sigma^* : (A \rightarrow w) \in P\}$$

$$prev := \emptyset$$

**while** ( $term \neq prev$ ):

$$prev := term$$

$$term := term \cup \{A \in V \mid \exists w \in (term \cup \Sigma)^* : (A \rightarrow w) \in P\}$$

## Algorithm: detecting of reachability (forwards)

$$reach := \{S\}$$

$$new := \{S\}$$

**while** ( $new \neq \emptyset$ ):

$$prev := reach$$

$$reach := reach \cup \{C \in V \mid \exists B \in new : (B \rightarrow w) \in P, C \in w\}$$

$$new := reach \setminus prev$$

# Detection of useless variables

## Algorithm: detection of termination (backwards)

$$term := \{A \in V \mid \exists w \in \Sigma^* : (A \rightarrow w) \in P\}$$

$$prev := \emptyset$$

**while** ( $term \neq prev$ ):

$$prev := term$$

$$term := term \cup \{A \in V \mid \exists w \in (term \cup \Sigma)^* : (A \rightarrow w) \in P\}$$

## Algorithm: detecting of reachability (forwards)

$$reach := \{S\}$$


$$new := \{S\}$$

**while** ( $new \neq \emptyset$ ):

$$prev := reach$$

$$reach := reach \cup \{C \in V \mid \exists B \in new : (B \rightarrow w) \in P, C \in w\}$$

$$new := reach \setminus prev$$

First clean up  $V \setminus term$ , then  $V' \setminus reach$  (why?) 

# Contents

- 1 Syntax of programming languages
  - Ambiguity
  - Naive Parsing
- 2 Chomsky normal form (CNF)
  - Grammar Transformations
  - CNF definition
  - Transformation to CNF
  - Optimisations
- 3 The CYK parsing-algorithm

# Cocke-Younger-Kasami (CYK) parsing

Starting point: Grammar in CNF; word  $w$  to be parsed

- Suppose that  $w = a_1 a_2 \cdots a_n$  (where  $n \geq 0$  and  $a_i \in V \cup \Sigma$ )
- Notation: sub-word  $w_{ij} = a_i \cdots a_j$  (with  $1 \leq i \leq j \leq n$ )

# Cocke-Younger-Kasami (CYK) parsing

**Starting point:** Grammar in CNF; word  $w$  to be parsed

- Suppose that  $w = a_1 a_2 \cdots a_n$  (where  $n \geq 0$  and  $a_i \in V \cup \Sigma$ )
- Notation: sub-word  $w_{ij} = a_i \cdots a_j$  (with  $1 \leq i \leq j \leq n$ )

**Basic idea:** **Dynamic Programming:**

Consider increasingly longer sub-words

- $X_{ij} \subseteq V$ : set of variables with a derivation to  $w_{ij}$
- These can be built step-by-step (iterate over  $j - i$ )

# Cocke-Younger-Kasami (CYK) parsing

**Starting point:** Grammar in CNF; word  $w$  to be parsed

- Suppose that  $w = a_1 a_2 \cdots a_n$  (where  $n \geq 0$  and  $a_i \in V \cup \Sigma$ )
- Notation: sub-word  $w_{ij} = a_i \cdots a_j$  (with  $1 \leq i \leq j \leq n$ )

**Basic idea:** **Dynamic Programming:**

Consider increasingly longer sub-words

- $X_{ij} \subseteq V$ : set of variables with a derivation to  $w_{ij}$
- These can be built step-by-step (iterate over  $j - i$ )

**Algorithm:** Bottom-up

**for**  $len$  **in range**(1,  $n$ ):

# Cocke-Younger-Kasami (CYK) parsing

**Starting point:** Grammar in CNF; word  $w$  to be parsed

- Suppose that  $w = a_1 a_2 \cdots a_n$  (where  $n \geq 0$  and  $a_i \in V \cup \Sigma$ )
- Notation: sub-word  $w_{ij} = a_i \cdots a_j$  (with  $1 \leq i \leq j \leq n$ )

**Basic idea:** **Dynamic Programming:**

Consider increasingly longer sub-words

- $X_{ij} \subseteq V$ : set of variables with a derivation to  $w_{ij}$
- These can be built step-by-step (iterate over  $j - i$ )

**Algorithm:** Bottom-up

```

for  $len$  in  $\text{range}(1, n)$ :
    for  $i$  in  $\text{range}(1, n - len + 1)$ :
         $j := i + len - 1$ 
  
```

# Cocke-Younger-Kasami (CYK) parsing

**Starting point:** Grammar in CNF; word  $w$  to be parsed

- Suppose that  $w = a_1 a_2 \cdots a_n$  (where  $n \geq 0$  and  $a_i \in V \cup \Sigma$ )
- Notation: sub-word  $w_{ij} = a_i \cdots a_j$  (with  $1 \leq i \leq j \leq n$ )

**Basic idea:** **Dynamic Programming:**

Consider increasingly longer sub-words

- $X_{ij} \subseteq V$ : set of variables with a derivation to  $w_{ij}$
- These can be built step-by-step (iterate over  $j - i$ )

**Algorithm:** Bottom-up

**for**  $len$  **in**  $range(1, n)$ :

**for**  $i$  **in**  $range(1, n - len + 1)$ :

$j := i + len - 1$

**if**  $len == 1$ :

$X_{ij} := \{A \mid (A \rightarrow a_{ij}) \in P\}$

**else:**

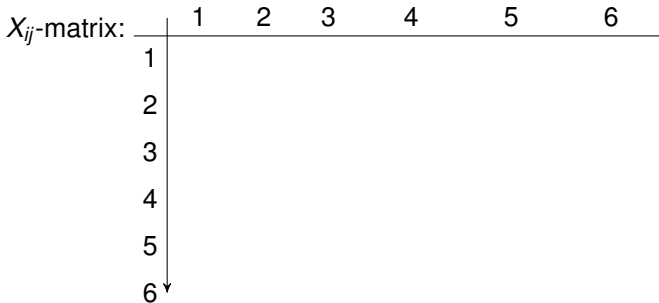
$X_{ij} := \{A \mid (A \rightarrow BC) \in P, B \in X_{ik}, C \in X_{k+1,j}\}$

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):  
 $S \rightarrow AT \mid AB$        $A \rightarrow a$   
 $T \rightarrow XB$                $B \rightarrow b$   
 $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :



# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):  
 $S \rightarrow AT \mid AB$        $A \rightarrow a$   
 $T \rightarrow XB$                $B \rightarrow b$   
 $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2						
3						
4						
5						
6						

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2		{A}				
3						
4						
5						
6						

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2		{A}				
3			{A}			
4						
5						
6						

# Example CYK parsing

- Grammar of  $\{a^i b^j \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2		{A}				
3			{A}			
4				{B}		
5						
6						

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2		{A}				
3			{A}			
4				{B}		
5					{B}	
6						

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}					
2		{A}				
3			{A}			
4				{B}		
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$				
2		{A}				
3			{A}			
4				{B}		
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$				
2		{A}	$\emptyset$			
3			{A}			
4				{B}		
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^j \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅				
2		{A}	∅			
3			{A}			
4				{B}		
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$				
2		{A}	$\emptyset$			
3			{A}	{S, X}		
4				{B}		
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^j \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅				
2		{A}	∅			
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^j \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅				
2		{A}	∅			
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$				
2		{A}	$\emptyset$			
3			{A}	{S, X}		
4				{B}	$\emptyset$	
5					{B}	$\emptyset$
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅				
2		{A}	∅			
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$	$\emptyset$			
2		{A}	$\emptyset$			
3			{A}	{S, X}		
4				{B}	$\emptyset$	
5					{B}	$\emptyset$
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$	$\emptyset$			
2		{A}	$\emptyset$			
3			{A}	{S, X}		
4				{B}	$\emptyset$	
5					{B}	$\emptyset$
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅			
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}		
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB \quad A \rightarrow a$
  - $T \rightarrow XB \quad B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB \quad A \rightarrow a$
  - $T \rightarrow XB \quad B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅			
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$	$\emptyset$			
2		{A}	$\emptyset$	$\emptyset$		
3			{A}	{S, X}	{T}	
4				{B}	$\emptyset$	$\emptyset$
5					{B}	$\emptyset$
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):
  - $S \rightarrow AT \mid AB$        $A \rightarrow a$
  - $T \rightarrow XB$                $B \rightarrow b$
  - $X \rightarrow AT \mid AB$
- Parsing of word  $w = \mathbf{a a a b b b}$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅		
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅		
2		{A}	∅	∅		
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅		
2		{A}	∅	∅	{S, X}	
3			{A}	{S, X}	{T}	
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aaabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$	$\emptyset$	$\emptyset$		
2		{A}	$\emptyset$	$\emptyset$	{S, X}	
3			{A}	{S, X}	{T}	$\emptyset$
4				{B}	$\emptyset$	$\emptyset$
5					{B}	$\emptyset$
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	∅
2		{A}	∅	∅	{S, X}	
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	
2		{A}	∅	∅	{S, X}	
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = aabbb$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

# Example CYK parsing

- Grammar of  $\{a^i b^i \mid i > 0\}$  (in Chomsky normal form):

$$S \rightarrow AT \mid AB \quad A \rightarrow a$$

$$T \rightarrow XB \quad B \rightarrow b$$

$$X \rightarrow AT \mid AB$$

- Parsing of word  $w = a a a b b b$ :

$X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	{S, X}
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

## Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing

# Derivation trees in the CYK-algorithm

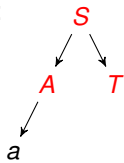
- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{S, X}
2		{A}	$\emptyset$	$\emptyset$	{S, X}	{T}
3			{A}	{S, X}	{T}	$\emptyset$
4				{B}	$\emptyset$	$\emptyset$
5					{B}	$\emptyset$
6						{B}

# Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

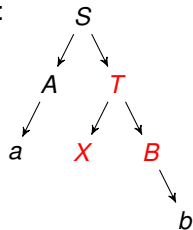
	1	2	3	4	5	6
1	{A, X}	∅	∅	∅	∅	{S, X}
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}



# Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A, X}	∅	∅	∅	∅	{S, X}
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

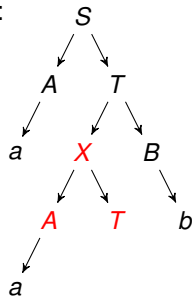


# Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A} →	∅	∅	∅	∅	{S, X}
2		{A} →	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

Red arrows in the original image indicate the path: (1,1) → (1,6) → (2,6) → (2,5) → (3,5) → (3,4) → (4,4) → (5,4) → (6,4).



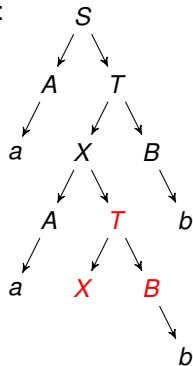
# Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

	1	2	3	4	5	6
1	{A}	∅	∅	∅	∅	{S, X}
2		{A}	∅	∅	{S, X}	{T}
3			{A}	{S, X}	{T}	∅
4				{B}	∅	∅
5					{B}	∅
6						{B}

Red arrows in the table indicate the backpointers used to reconstruct the derivation tree:

- From (1,6) to (1,1) and (1,6) to (1,6)
- From (2,6) to (2,2) and (2,6) to (2,6)
- From (3,5) to (3,3) and (3,5) to (3,5)
- From (3,4) to (3,4) and (3,4) to (3,5)
- From (5,5) to (5,5) and (5,5) to (6,6)



# Derivation trees in the CYK-algorithm

- The CYK-algorithm decides whether a word is in the language
- This is *not* yet the same as parsing
- The derivation tree is implicitly in the  $X_{ij}$ -matrix:

