COM S 342

Recitation 09/09/2019 – 09/11/2019

Topic

- Context free grammar:
  - Derivation,
  - Parse tree,
  - Grammar design,
  - Remove ambiguity.

Grammar (review)

- Specification of legal strings in a programming language.
- Key elements:
  - Production rule, non-terminal, terminal.
    - What symbols can be expanded?
    - Backus Naur Form (BNF)
    - * means zero or more, + means one or more, | defines alternatives
  - Syntax derivation: leftmost, rightmost
    - Systematic proof that a string belongs to a language
    - Replace exactly one symbol in a single step

Derivation

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Write leftmost and rightmost derivation of the string "babbab".

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

Derivation

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Write leftmost and rightmost derivation of the string "babbab".

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

Leftmost Derivation: At each derivation point, the leftmost non-terminal is expanded.

Derivation

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Write leftmost and rightmost derivation of the string “babbab”.

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

S → AA → bAA → bAbA → babA → babbA

→ babbAb → babbab

Derivation

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Write leftmost and rightmost derivation of the string “babbab”.

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

Rightmost Derivation: At each derivation point, the rightmost non-terminal is expanded.

Derivation

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Write leftmost and rightmost derivation of the string “babbab”.

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

S → AA → AAb → Aab → Abab → Abbab

→ bAbbab → babbab

Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol).

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

A parse tree results from the derivation sequence.

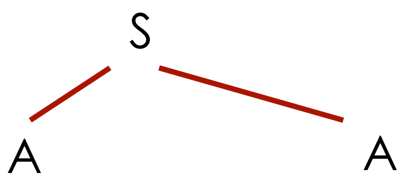
- Each node in the tree is a terminal or non-terminal in the production rule.
- Each edge in the tree from a non-terminal results from the application of production rule on the non-terminal.
- Application of production rule always result in new nodes in the tree.
- A terminal is a leaf node

Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

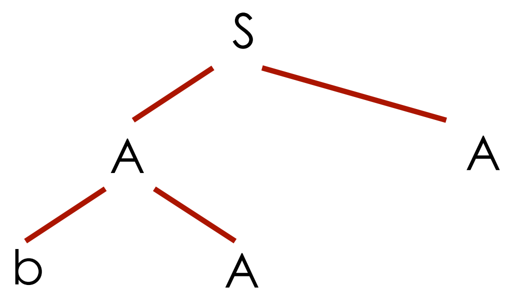


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

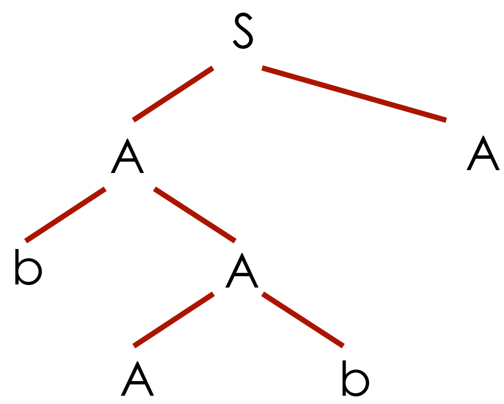


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

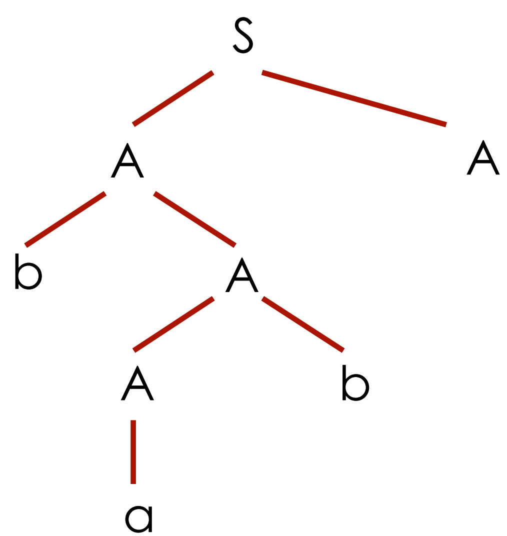


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

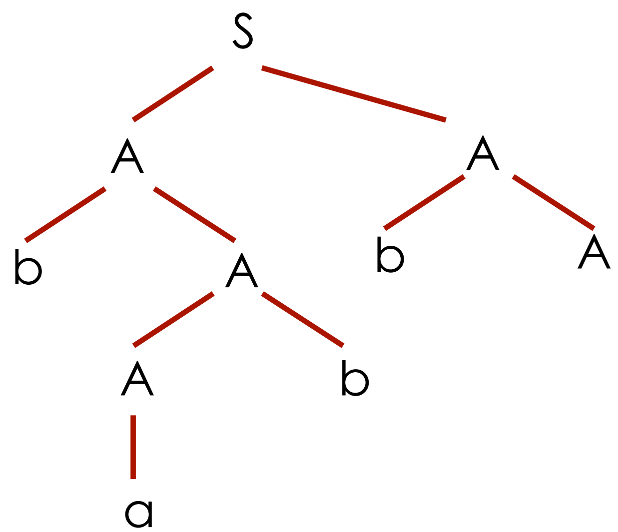


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$

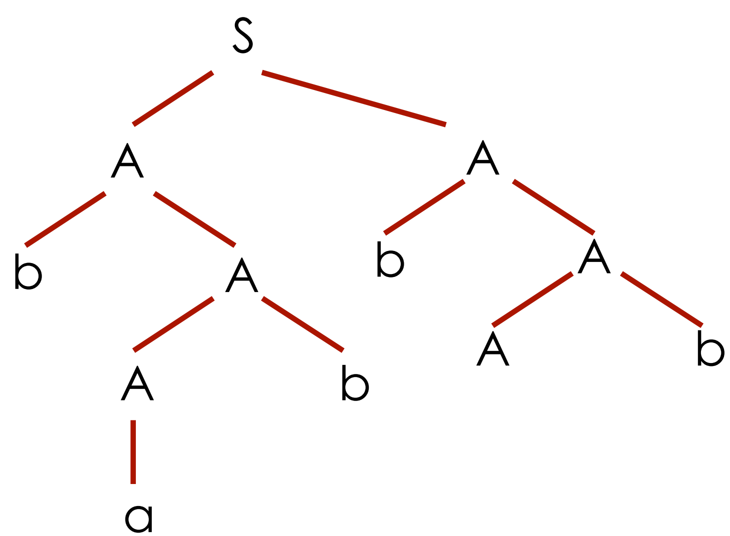


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string "babbab".

$$
\begin{aligned}
S &\to AA \\
A &\to AAA \\
A &\to a \\
A &\to bA \\
A &\to Ab
\end{aligned}
$$

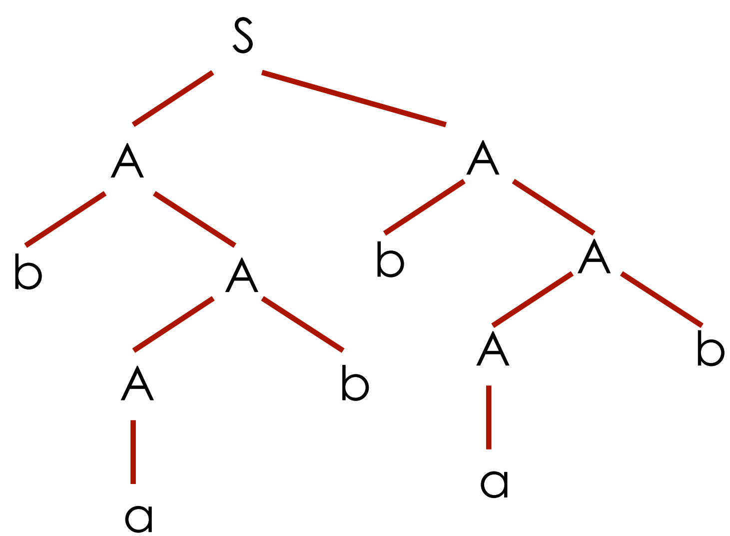


Parse Tree

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol). Draw a parse tree for the string “babbab”.

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$



Ambiguity

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol).

$$
\begin{aligned}
S &\to AA \\
A &\to AAA \\
A &\to a \\
A &\to bA \\
A &\to Ab
\end{aligned}
$$

Prove or disprove the grammar is ambiguous.

Ambiguity

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol).

$$
\begin{aligned}
S &\rightarrow AA \\
A &\rightarrow AAA \\
A &\rightarrow a \\
A &\rightarrow bA \\
A &\rightarrow Ab
\end{aligned}
$$

Prove or disprove the grammar is ambiguous.

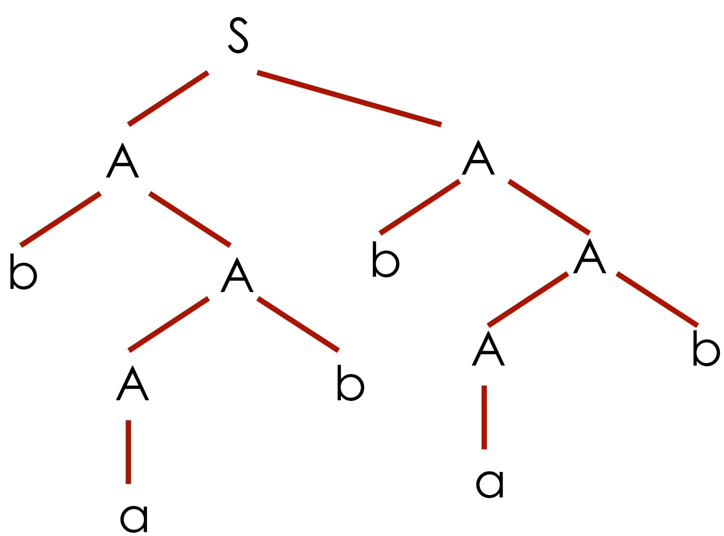
- A grammar is *ambiguous* if there exists at least two distinct parse trees for the derivation of the same string.

Ambiguity

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol).

$$S \rightarrow AA$$
$$A \rightarrow AAA$$
$$A \rightarrow a$$
$$A \rightarrow bA$$
$$A \rightarrow Ab$$

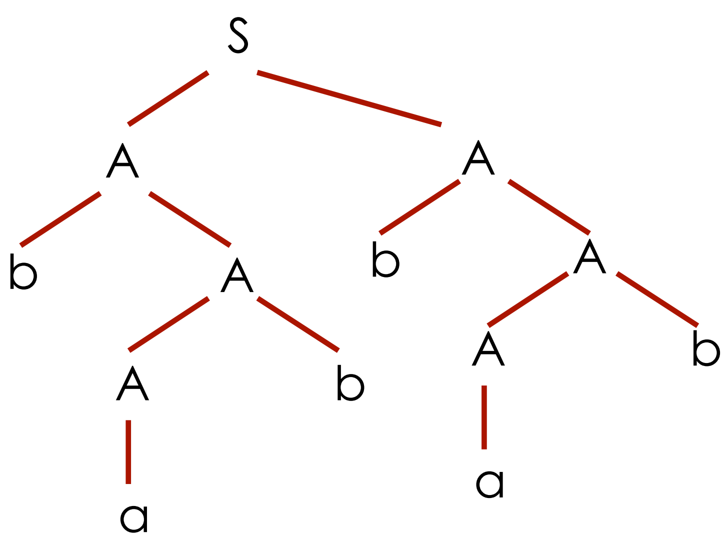


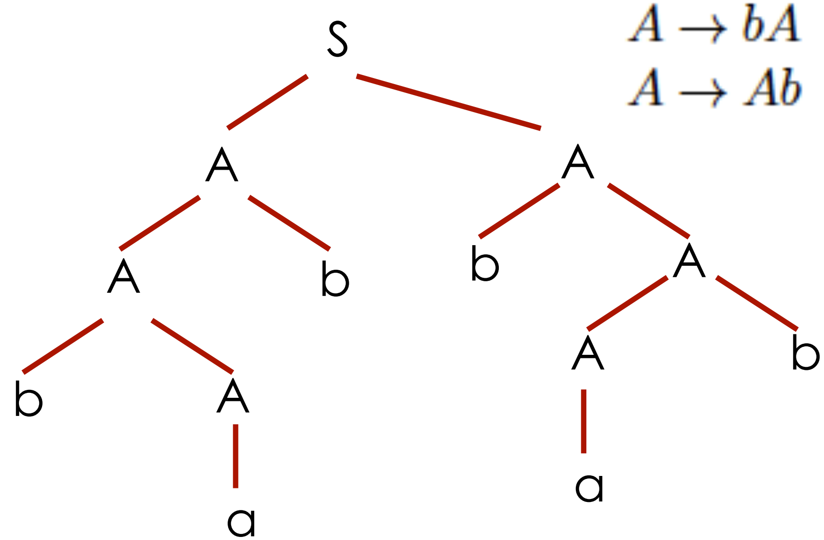
Ambiguity

- Grammar:

Consider the following production rules of a grammar G (all lower-case letters are terminals and S is the start symbol).

$$S \to AA$$
$$A \to AAA$$
$$A \to a$$
$$A \to bA$$
$$A \to Ab$$





Ambiguous!

Ambiguity

Remove ambiguity:

- Add delimiters (e.g., parenthesis; begin and end in if statements)
- Add operator precedence and associativity

Operator Precedence:

$S \rightarrow S \mathbin{@} S \mid S \# S \mid b$

Example:

$b \mathbin{@} b \# b \mathbin{@} b$

Ambiguity

Remove ambiguity:

Operator Precedence:

- If more than one operator is present in the expression, the precedence order decides the order in which the operators should be applied.
- Add non-terminals for each precedence level. Push the higher levels towards the bottom of the parse-tree (stratification of tree)

$S \rightarrow S \mathbin{@} S \mid S \# S \mid b$

Example:

b @ b # b @ b

Ambiguity

Remove ambiguity:

Operator Precedence:

- If more than one operator is present in the expression, the precedence order decides the order in which the operators should be applied.
- Add non-terminals for each precedence level. Push the higher levels towards the bottom of the parse-tree (stratification of tree)

$S \rightarrow S\ @\ S \mid S\ \#\ S \mid b$

Example:

b @ b # b @ b

$S \rightarrow S\ @\ S \mid A$
$A \rightarrow A\ \#\ A \mid b$

Example:

b @ b # b @ b

Ambiguity

Remove ambiguity:

Associativity:

If the same operator appears more than once in the same expression, then associativity rule decides the order in which the operators should be applied.

Ambiguity

Remove ambiguity:

Associativity:

$$
\begin{aligned}
S &\longrightarrow S + S \mid S - S \mid T \\
T &\longrightarrow T * T \mid T / T \mid \mathtt{part}
\end{aligned}
$$

Example string: 3 + 4 + 2.

The grammar allows two different derivation trees for the string **3 + 4 + 2**, one corresponding to the structure **(3 + 4) + 2** and one corresponding to the structure **3 + (4 + 2)**.

Ambiguity

Remove ambiguity:

Associativity:

$$\begin{array}{lcl} S & \longrightarrow & S + S \mid S - S \mid T \\ T & \longrightarrow & T * T \mid T / T \mid \mathtt{part} \end{array}$$

For the op in question,

We can impose left-associativity (resp. right-associativity) by using a left-recursive (resp. right-recursive) production

$$\begin{array}{lcl} S & \longrightarrow & S + T \mid S - S \mid T \\ T & \longrightarrow & T * T \mid T / T \mid \mathtt{part} \end{array}$$

Designing a Grammar

Design a grammar that allows (a|b)*abb

Designing a Grammar

Design a grammar that allows (a|b)*abb

<u>Example accepted strings:</u>

abb

babb

baabb

ababb

Designing a Grammar

Design a grammar that allows (a|b)*abb

$S \rightarrow aS \mid bS \mid aA_1$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow bA_3$

$A_3 \rightarrow \varepsilon$