

Foundations of Computing

Lecture 9

Arkady Yerukhimovich

February 11, 2025

Outline

- 1 Midterm 1 Announcement
- 2 Lecture 8 Review
- 3 Grammars
- 4 Designing Context-Free Grammars
- 5 Derivations and Parse Trees

Midterm 1 – February 20

- Exam 1 will be in class on February 20 (next Thursday)
- It will cover NFA/DFA/regular languages, and PDAs/Context-free grammars

Exam Policies

- The exam will be closed book and closed notes
- You will be allowed two 8.5×11 pieces of paper with notes – anything you choose
- No computers, calculators, or other digital devices – bring a pencil or pen

Important

If you have a conflict with this exam, let me know ASAP!

Next Week

- Lecture and lab next week will be largely for review
- This is your chance to clear things up before the midterm

Bring your questions!

Outline

- 1 Midterm 1 Announcement
- 2 **Lecture 8 Review**
- 3 Grammars
- 4 Designing Context-Free Grammars
- 5 Derivations and Parse Trees

- Pushdown Automata
 - Using a stack to recognize non-regular languages
 - Examples of building PDAs

- Pushdown Automata
 - Using a stack to recognize non-regular languages
 - Examples of building PDAs

Today

An alternative formulation for languages accepted by PDAs

Exercise – Work in Groups

Show a PDA that recognizes the language

$$L = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

- 1 Describe a PDA algorithm for this language
- 2 Write the states and transition function
- 3 Draw the PDA graph



Exercise – Work in Groups

Show a PDA that recognizes the language

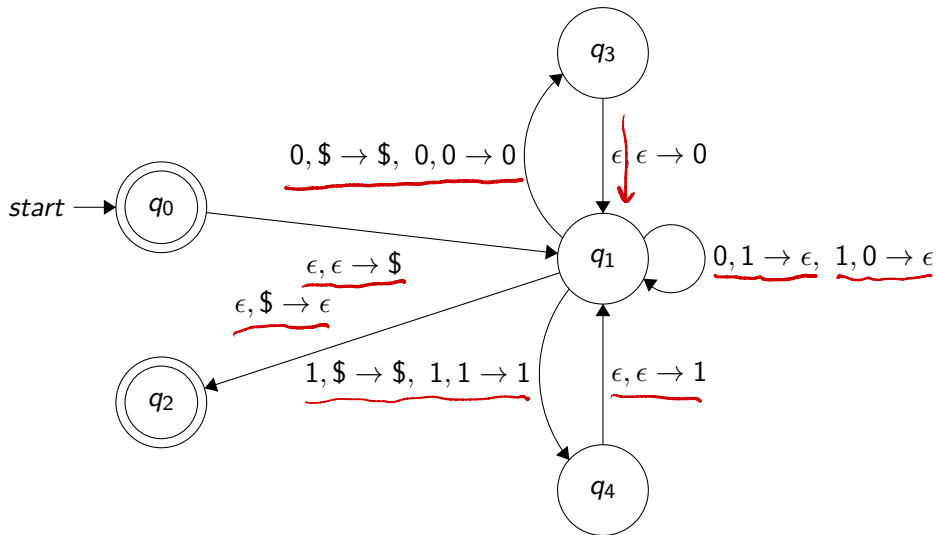
$$L = \{w \mid w \text{ has an equal number of 0s and 1s}\}$$

- ① Describe a PDA algorithm for this language
- ② Write the states and transition function
- ③ Draw the PDA graph

Solution:

- ① Push \$ on the stack
- ② If input is 0, pop value from the stack
 - If it's a 0 or \$ push it back on the stack and push another 0 on top
 - If it's a 1 pop it off the stack 
- ③ If input is 1, pop value from the stack
 - If it's a 1 or \$ push it back and push another 1 on top 
 - If it's a 0 pop it off the stack
- ④ When the input is done, if \$ is top of the stack, accept

Exercise – Work in Groups



Outline

- 1 Midterm 1 Announcement
- 2 Lecture 8 Review
- 3 Grammars**
- 4 Designing Context-Free Grammars
- 5 Derivations and Parse Trees

Representing Languages

Recall that a language L is a set of strings

We have seen several ways for describing a language L :

- DFA/NFA – the language of strings accepted by M
- Regular expressions
- Pushdown Automata

Representing Languages

Recall that a language L is a set of strings

We have seen several ways for describing a language L :

- DFA/NFA – the language of strings accepted by M
- Regular expressions
- Pushdown Automata

Grammars

- A grammar is a set of rules by which strings in L are constructed/derived

Representing Languages

Recall that a language L is a set of strings

We have seen several ways for describing a language L :

- DFA/NFA – the language of strings accepted by M
- Regular expressions
- Pushdown Automata

Grammars

- A grammar is a set of rules by which strings in L are constructed/derived
- Today, we will focus on context-free grammars and the languages they represent

A grammar G consists of:

- V – finite set of variables (usually Capital Letters)
- Σ – a finite set of symbols called the terminals (usually lower case letters)
- R – finite set of rules how strings in L can be produced
- $S \in V$ – start variable

If no S is specified, can assume it is the variable in the first rule.

A grammar G consists of:

- V – finite set of variables (usually Capital Letters)
- Σ – a finite set of symbols called the terminals (usually lower case letters)
- R – finite set of rules how strings in L can be produced
- $S \in V$ – start variable

If no S is specified, can assume it is the variable in the first rule.

Definition

For a grammar G , the language L_G generated by G is the set of all terminal strings that can be produced by G starting with the start symbol by using a sequence of the production rules.

Example 1

Consider the following grammar G_1 :

- $V = \{A, B\}$
- $\Sigma = \{0, 1, \#\}$
- $R =$

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

- $S = A$

Example 1

Consider the following grammar G_1 :

- $V = \{A, B\}$
- $\Sigma = \{0, 1, \#\}$
- $R =$

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

- $S = A$

Strings Produced by G_1 :

Example 1

Consider the following grammar G_1 :

- $V = \{A, B\}$
- $\Sigma = \{0, 1, \#\}$
- $R =$

1. $A \rightarrow 0A1$

2. $A \rightarrow B$

3. $B \rightarrow \#$

- $S = A$

Strings Produced by G_1 :

$$\underline{A} \Rightarrow \underline{0A1} \Rightarrow \underline{00A11} \Rightarrow \underline{000A111} \Rightarrow \underline{000B111} \Rightarrow \underline{000\#111}$$

Example 1

Consider the following grammar G_1 :

- $V = \{A, B\}$
- $\Sigma = \{0, 1, \#\}$
- $R =$

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

$$A \rightarrow 0A1 \mid B$$

- $S = A$

Strings Produced by G_1 :

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

$$L(G_1) = \{0^n \# 1^n\}$$

$$\{0^n \# 1^n \mid n \geq 0\}$$

Strings Produced by a Grammar

For a grammar G generating language L , can generate each string $w \in L$ as follows:

Strings Produced by a Grammar

For a grammar G generating language L , can generate each string $w \in L$ as follows:

- 1 Write down the start variable

Strings Produced by a Grammar

For a grammar G generating language L , can generate each string $w \in L$ as follows:

- 1 Write down the start variable
- 2 Find a written-down variable and a rule starting with that variable.
Replace the written variable with the right side of that rule

Strings Produced by a Grammar

For a grammar G generating language L , can generate each string $w \in L$ as follows:

- 1 Write down the start variable
- 2 Find a written-down variable and a rule starting with that variable.
Replace the written variable with the right side of that rule
- 3 Repeat Step 2 until no variables remain

Strings Produced by a Grammar

For a grammar G generating language L , can generate each string $w \in L$ as follows:

- 1 Write down the start variable
- 2 Find a written-down variable and a rule starting with that variable.
Replace the written variable with the right side of that rule
- 3 Repeat Step 2 until no variables remain

Rules Notation

- Rules can have multiple options separated by $|$ to indicate OR
- ϵ - empty string

Context-Free Grammars (CFG)

Definition

A grammar G is context-free if for all of its rules, the left side consists of exactly one variable and no terminals.

$$A \rightarrow 0A1$$

$$S \rightarrow$$

Context-Free Grammars (CFG)

Definition

A grammar G is context-free if for all of its rules, the left side consists of exactly one variable and no terminals.

- This is called context-free since a variable (on left side of rule) always produces same output, regardless of “context”

Context-Free Grammars (CFG)

Definition

A grammar G is context-free if for all of its rules, the left side consists of exactly one variable and no terminals.

- This is called context-free since a variable (on left side of rule) always produces same output, regardless of “context”
- Context-free grammars originated in the study of human languages
- They capture recursive structures common in language (e.g., noun phrases can be made of verb-phrases and vice-versa)
- Also, very useful for describing programming languages:
 - Can capture matching, nested brackets:

```
if x > 3 {  
    if y < 5 {  
        Do something  
    }  
}
```

Outline

- 1 Midterm 1 Announcement
- 2 Lecture 8 Review
- 3 Grammars
- 4 Designing Context-Free Grammars**
- 5 Derivations and Parse Trees

How to Design CFGs for L

Designing CFGs

- CFGs are inherently recursive (e.g., $A \rightarrow 0A1$) – need to think what happens when we recurse

How to Design CFGs for L

Designing CFGs

- CFGs are inherently recursive (e.g., $A \rightarrow 0A1$) – need to think what happens when we recurse
- Build a string from outside in

$$A \rightarrow 0 \underbrace{A}_{} 1 \rightarrow 0 \underbrace{0 A 1}_{} 1$$

How to Design CFGs for L

Designing CFGs

- CFGs are inherently recursive (e.g., $A \rightarrow 0A1$) – need to think what happens when we recurse
- Build a string from outside in
- Build from both ends at the same time (due to recursion)

How to Design CFGs for L

Designing CFGs

- CFGs are inherently recursive (e.g., $A \rightarrow 0A1$) – need to think what happens when we recurse
- Build a string from outside in
- Build from both ends at the same time (due to recursion)

This is Tricky

Designing CFGs is not natural, takes lots of practice

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

- 1 Build a grammar for $L_1 = \{0^n 1^n \mid n \geq 0\}$

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

- 1 Build a grammar for $L_1 = \{0^n 1^n \mid n \geq 0\}$

$$S_1 \rightarrow \underline{0S_11} \mid \epsilon$$

$00\dots0 \times 11\dots1$

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

- 1 Build a grammar for $L_1 = \{0^n 1^n \mid n \geq 0\}$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

- 2 Build a grammar for $L_2 = \{1^n 0^n \mid n \geq 0\}$

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

- 1 Build a grammar for $L_1 = \{0^n 1^n \mid n \geq 0\}$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

- 2 Build a grammar for $L_2 = \{1^n 0^n \mid n \geq 0\}$

$$S_2 \rightarrow 1S_20 \mid \epsilon$$

Example 1

Question

Design a CFG for the language $L = \{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

Intuition: Generate each of the two, and take the union

- 1 Build a grammar for $L_1 = \{0^n 1^n \mid n \geq 0\}$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

- 2 Build a grammar for $L_2 = \{1^n 0^n \mid n \geq 0\}$

$$S_2 \rightarrow 1S_20 \mid \epsilon$$

- 3 Combine the two to give the grammar for the union

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

$$S_2 \rightarrow 1S_20 \mid \epsilon$$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

$$\underline{a^{n-n}} \parallel a^n b^n$$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

- 1 Build a grammar for $L_1 = \{a^n b^n \mid n \geq 0\}$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

- 1 Build a grammar for $L_1 = \{a^n b^n \mid n \geq 0\}$

$$C \rightarrow aCb \mid \epsilon$$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

- 1 Build a grammar for $L_1 = \{a^n b^n \mid n \geq 0\}$

$$C \rightarrow aCb \mid \epsilon$$

- 2 Build a grammar for $L_2 = \{a^{m-n} \mid m > n \geq 0\}$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

- 1 Build a grammar for $L_1 = \{a^n b^n \mid n \geq 0\}$

$$C \rightarrow aCb \mid \epsilon$$

- 2 Build a grammar for $L_2 = \{a^{m-n} \mid m > n \geq 0\}$

$$A \rightarrow \underline{aA} \mid \underline{a}$$

Example 2

Question

Design a CFG for the language $L = \{a^m b^n \mid m > n \geq 0\}$

Intuition: This is concatenation of $a^n b^n$ and a^{m-n}

- ① Build a grammar for $L_1 = \{a^n b^n \mid n \geq 0\}$

$a^n b^n$

$$\underline{C \rightarrow aCb \mid \epsilon}$$

aab
 $aaabbb$

~~$aaabbb$~~

- ② Build a grammar for $L_2 = \{a^{m-n} \mid m > n \geq 0\}$

$$A \rightarrow \underline{aA} \mid \underline{a}$$

- ③ Concatenate the two to give the grammar for L

$$\boxed{S \rightarrow AC}$$

$$C \rightarrow aCb \mid \epsilon$$

$$A \rightarrow aA \mid a$$

$$S \rightarrow AC \Rightarrow$$

$$AaCb \Rightarrow Aab \Rightarrow$$

$$\Rightarrow aab$$

Exercise

Give a CFG for $L = \{a^m b^n \mid m \neq n, m, n \geq 0\}$

Hint: Think of this as the union of two languages

Outline

- 1 Midterm 1 Announcement
- 2 Lecture 8 Review
- 3 Grammars
- 4 Designing Context-Free Grammars
- 5 Derivations and Parse Trees

Derivations

- Derivations of a string w in CFG G is the sequence of substitutions resulting in w

Derivations

- Derivations of a string w in CFG G is the sequence of substitutions resulting in w

Consider Grammar G_1

$$R = A \rightarrow 0A1 \mid B, \quad B \rightarrow \#$$

Derivations

- Derivations of a string w in CFG G is the sequence of substitutions resulting in w

Consider Grammar G_1

$$R = A \rightarrow 0A1 \mid B, \quad B \rightarrow \#$$

- Find derivation of $w = 000\#111$


Derivations

- Derivations of a string w in CFG G is the sequence of substitutions resulting in w

Consider Grammar G_1

$$R = A \rightarrow 0A1 \mid B, \quad B \rightarrow \#$$

- Find derivation of $w = 000\#111$

$$\textcolor{red}{A} \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$


Parse Trees

- Parse trees are trees with nodes labeled by symbols of a CFG G

Parse Trees

- Parse trees are trees with nodes labeled by symbols of a CFG G
- Leaves: Labeled by a terminal or ϵ
 - These labels read left-to-right give you the string represented by this parse tree

Parse Trees

- Parse trees are trees with nodes labeled by symbols of a CFG G
- Leaves: Labeled by a terminal or ϵ
 - These labels read left-to-right give you the string represented by this parse tree
- Interior nodes: Labeled by a variable (on left-hand side of a production rule)
 - Children are labeled by right-hand side of parent's rule

Parse Trees

- Parse trees are trees with nodes labeled by symbols of a CFG G
- Leaves: Labeled by a terminal or ϵ
 - These labels read left-to-right give you the string represented by this parse tree
- Interior nodes: Labeled by a variable (on left-hand side of a production rule)
 - Children are labeled by right-hand side of parent's rule
- Root: Labeled by start variable

Parse Trees

- Parse trees are trees with nodes labeled by symbols of a CFG G
- Leaves: Labeled by a terminal or ϵ
 - These labels read left-to-right give you the string represented by this parse tree
- Interior nodes: Labeled by a variable (on left-hand side of a production rule)
 - Children are labeled by right-hand side of parent's rule
- Root: Labeled by start variable

Why study parse trees?

- Parse trees help us understand the “meaning” of a string
- Also, how parsers can parse a string according to a grammar (e.g., of a programming language)

Parse Trees – An Example

Consider Grammar G_1

$$R = A \rightarrow 0A1 \mid B, \quad B \rightarrow \#$$

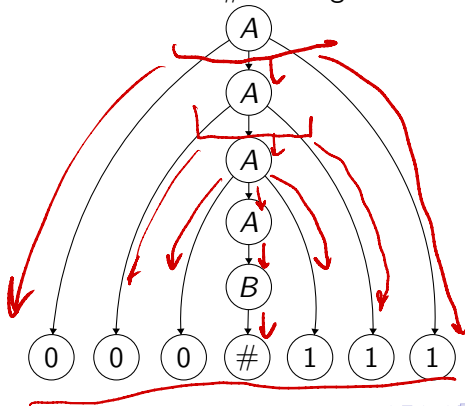
Parse tree for $000\#111$ in grammar G_1

Parse Trees – An Example

Consider Grammar G_1

$$R = A \rightarrow 0A1 \mid \underline{B}, \quad B \rightarrow \#$$

Parse tree for $000\#111$ in grammar G_1



Another Example

A Grammar G_2 for Arithmetic Statements

- $V = \{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$
- $R =$
 - $\langle \text{EXPR} \rangle \rightarrow \underline{\langle \text{EXPR} \rangle + \langle \text{TERM} \rangle} \mid \langle \text{TERM} \rangle$
 - $\langle \text{TERM} \rangle \rightarrow \underline{\langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle} \mid \langle \text{FACTOR} \rangle$
 - $\langle \text{FACTOR} \rangle \rightarrow \underline{(\langle \text{EXPR} \rangle)} \mid \underline{a}$

Another Example

A Grammar G_2 for Arithmetic Statements

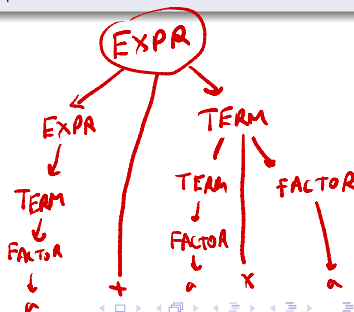
- $V = \{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$
- $R =$
 $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle$
 $\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle$
 $\langle \text{FACTOR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \mid a$
- What is $L(G_2)$? $a + (a \times a)$

Another Example

A Grammar G_2 for Arithmetic Statements

- $V = \{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$
- $R =$
 $\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle$
 $\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle$
 $\langle \text{FACTOR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \mid a$

- What is $L(G_2)$?
- Parse tree for $a + a \times a$



Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced

Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced
- 2 A string w is derived *ambiguously* if it has two or more different leftmost derivations

Ambiguity of Grammars

Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced
- 2 A string w is derived *ambiguously* if it has two or more different leftmost derivations
- 3 Grammar G is ambiguous if it generates some string ambiguously

Ambiguity of Grammars

Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced
- 2 A string w is derived *ambiguously* if it has two or more different leftmost derivations
- 3 Grammar G is ambiguous if it generates some string ambiguously
- 4 A language L is *inherently ambiguous* if it can only be generated by ambiguous grammars

Ambiguity of Grammars

Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced
- 2 A string w is derived *ambiguously* if it has two or more different leftmost derivations
- 3 Grammar G is ambiguous if it generates some string ambiguously
- 4 A language L is *inherently ambiguous* if it can only be generated by ambiguous grammars

Is ambiguity a problem?

- Ambiguous derivation may lead to different meanings for the string
Example: The girl touches the boy with the flower

Ambiguity of Grammars

Definitions

- 1 A derivation of w in G is a *leftmost derivation* if at every step, the leftmost variable is replaced
- 2 A string w is derived *ambiguously* if it has two or more different leftmost derivations
- 3 Grammar G is ambiguous if it generates some string ambiguously
- 4 A language L is *inherently ambiguous* if it can only be generated by ambiguous grammars

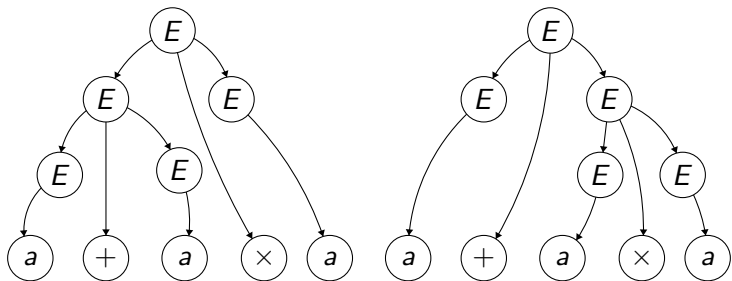
Is ambiguity a problem?

- Ambiguous derivation may lead to different meanings for the string
Example: The girl touches the boy with the flower
- Unfortunately, ambiguous languages cannot be made unambiguous

An Example

Consider the following grammar G_3

$$E \rightarrow E + E \mid E \times E \mid (E) \mid a$$



Two parse trees for the string $a + a \times a$

- Equivalence between CFGs and PDAs
- A pumping lemma for CFGs