# Foundations of Computing
## Lecture 9

Arkady Yerukhimovich

February 14, 2023

# Outline

# Midterm 1 – February 23

- Exam 1 will be in class on February 23 (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 $8.5 \times 11$ 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

# Lecture 8 Review

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

# Lecture 8 Review

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

## Today

An alternative formulation for languages accepted by PDAs

# Outline

# 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

## Grammar

A grammar $G$ consists of:

- $V$ – finite set of variables (usually Capital Letters)
- $\Sigma$ – 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.

# Grammar

A grammar $G$ consists of:

- $V$ – finite set of variables (usually Capital Letters)
- $\Sigma$ – 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 $G_1$:

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

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

- $S = A$

# Example 1

### Consider the following grammar $G_1$:

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

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

- $S = A$

Strings Produced by $G_1$:

## Example 1

### Consider the following grammar $G_1$:

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

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

- $S = A$

Strings Produced by $G_1$:

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

## Example 1

### Consider the following grammar $G_1$:

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

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

- $S = A$

Strings Produced by $G_1$:

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

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

# Strings Produced by a Grammar

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

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

# 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.

# 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)
  - a girl with a flower likes the boy

# 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)
  - a girl with a flower likes the boy
- Also, very useful for describing programming languages:
  - Can capture matching, nested brackets:
    if $x > 3$ {
        if $y < 5$ {
          Do something
        }
    }

# Outline

# Parse Trees aka. Derivation Trees

# Parse Trees aka. Derivation Trees

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

# Parse Trees aka. Derivation Trees

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

# Parse Trees aka. Derivation Trees

- Parse trees are trees with nodes labeled by symbols of a CFG $G$
- Leaves: Labeled by a terminal or $\epsilon$
  - 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 aka. Derivation Trees

- Parse trees are trees with nodes labeled by symbols of a CFG *G*
- Leaves: Labeled by a terminal or $\epsilon$
    - 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 aka. Derivation Trees

- Parse trees are trees with nodes labeled by symbols of a CFG *G*
- Leaves: Labeled by a terminal or $\epsilon$
    - 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

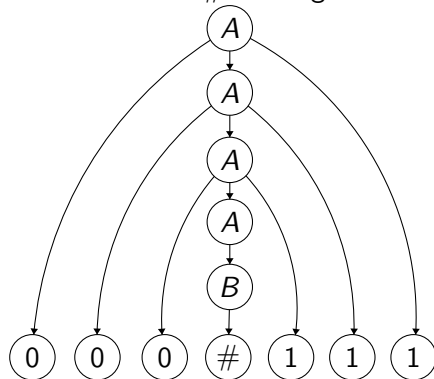## Recall Grammar $G_1$

$$R = A \to 0A1 \mid B, \qquad B \to \#$$

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

# Parse Trees – An Example

## Recall Grammar $G_1$

$$R = A \to 0A1 \mid B, \qquad B \to \#$$

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

# Another Example

## A Grammar $G_2$ for Arithmetic Statements

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

# Another Example

## A Grammar $G_2$ for Arithmetic Statements

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

$$\begin{aligned}
\langle EXPR \rangle &\rightarrow \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle \\
\langle TERM \rangle &\rightarrow \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle \\
\langle FACTOR \rangle &\rightarrow (\langle EXPR \rangle) \mid a
\end{aligned}$$

- What is $L(G_2)$?

# Another Example

## A Grammar $G_2$ for Arithmetic Statements

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

$$\begin{aligned}
\langle EXPR \rangle &\rightarrow \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle \\
\langle TERM \rangle &\rightarrow \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle \\
\langle FACTOR \rangle &\rightarrow (\langle EXPR \rangle) \mid a
\end{aligned}$$

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

# Ambiguity of Grammars

## Definitions

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

# 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

# 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
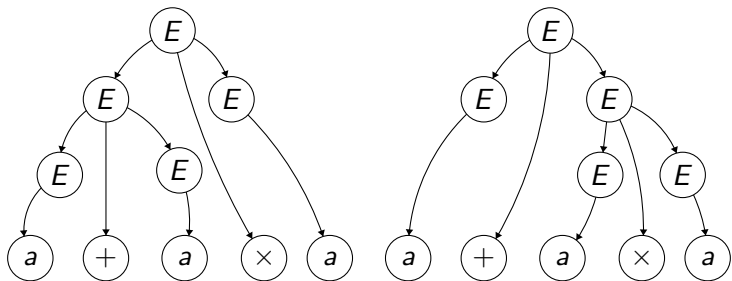
## Is ambiguity a problem?

- 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 $a + a \times a$

# Outline

## Designing CFGs

## Designing CFGs

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

## 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

# 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)

## Designing CFGs

- CFGs are inherently recursive (e.g., $A \to 0A1$) – 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 $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

# 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 $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

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

1. Build a grammar for $L_1 = \{0^n1^n \mid n \geq 0\}$

$$S_1 \to 0S_11 \mid \epsilon$$

# Example 1

### Question

Design a CFG for the language $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

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

1. Build a grammar for $L_1 = \{0^n1^n \mid n \geq 0\}$

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

2. Build a grammar for $L_2 = \{1^n0^n \mid n \geq 0\}$

# Example 1

## Question

Design a CFG for the language $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

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

1. Build a grammar for $L_1 = \{0^n1^n \mid n \geq 0\}$

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

2. Build a grammar for $L_2 = \{1^n0^n \mid n \geq 0\}$

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

# Example 1

## Question

Design a CFG for the language $L = \{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

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

1. Build a grammar for $L_1 = \{0^n1^n \mid n \geq 0\}$

$$S_1 \to 0S_11 \mid \epsilon$$

2. Build a grammar for $L_2 = \{1^n0^n \mid n \geq 0\}$

$$S_2 \to 1S_20 \mid \epsilon$$

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

$$
\begin{aligned}
S &\to S_1 \mid S_2 \\
S_1 &\to 0S_11 \mid \epsilon \\
S_2 &\to 1S_20 \mid \epsilon
\end{aligned}
$$

# 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 $a^{m-n}$

# 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. $S \rightarrow AC$ – concatenate the two languages
2. $C \rightarrow aCb \mid \epsilon$ – $C$ generates $a^n b^n$
3. $A \rightarrow aA \mid a$ – $A$ generates at least one $a$

# 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. $S \rightarrow AC$ – concatenate the two languages
2. $C \rightarrow aCb \mid \epsilon$ – $C$ generates $a^n b^n$
3. $A \rightarrow aA \mid a$ – $A$ generates at least one $a$

Exercise: Show a parse tree for $w = aaab$

## Exercises

1. Consider grammar $G$: $S \rightarrow aSa \mid bSb \mid \epsilon$
   Show parse tree for *abbbba*

2. 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

# On Thursday

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