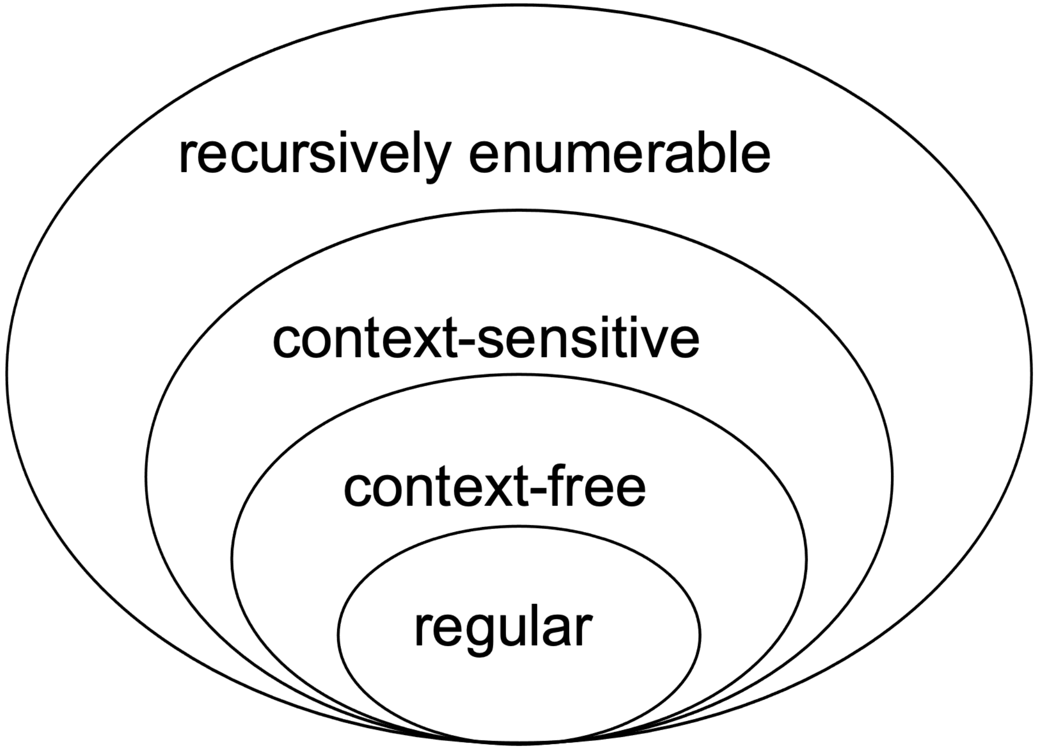
Introduction to Context Free Grammar

CPSC 501 Presentation

Jerry Wang

What we have learnt so far…
Classes of Formal Grammars

- **Chomsky Hierarchy[1]: 4 types of grammars**
  - Type 0: Turing-recognizable Languages - Turing Machine
  - Type 1: Context-sensitive Languages – Linear-Bounded Automaton
  - Type 2: Context-free Languages – Pushdown Automaton
  - Type 3: Regular Languages – Finite State Automaton



- **A simple language:** $\mathbf{L = \{0^n 1^n \mid n \geq 0\}}$
  - Not regular but Turing-recognizable
  - Also, context-free!

[1] Noam Chomsky (1956). "Three models for the description of language". IRE Transactions on Information Theory.

Context Free Grammar
Today's Outline

- **The Basics**
  - Syntax, Formal Definition, Derivations
- **Parsing Part 1**
  - A quick look at the Pushdown Automaton
  - CYK Algorithm: Can a string be generated from this grammar?
- **Parsing Part 2**
  - Top-down Parsing
  - Bottom-up Parsing
- **Summary**

The Basics
First Glance

- **An example context free grammar:**
  - $S \rightarrow 0S1$
  - $S \rightarrow \epsilon$

The Basics
Grammar Components

- **An example context free grammar:**
  - $S \to 0S1$
  - $S \to \epsilon$
- **Contains a collection of *production rules***
  - Also called *substitution rules* or *rewrite rules*.

The Basics
Grammar Components

- **An example context free grammar:**
  - $\boldsymbol{S} \rightarrow 0S1$
  - $\boldsymbol{S} \rightarrow \epsilon$
- **Each production rule ($\boldsymbol{V} \rightarrow \boldsymbol{w}$) contains…**
  - **A symbol called variable or non-terminals**
  - A right arrow
  - A string that contains other variables and terminals

The Basics
Grammar Components

- **An example context free grammar:**
  - $S \rightarrow 0S1$
  - $S \rightarrow \epsilon$
- **Each production rule ($V \rightarrow w$) contains…**
  - A symbol called variable or non-terminals
  - **A right arrow**
  - A string that contains other variables and terminals

The Basics
Grammar Components

- **An example context free grammar:**
  - $S \rightarrow \mathbf{0S1}$
  - $S \rightarrow \boldsymbol{\epsilon}$
- **Each production rule ($V \rightarrow w$) contains…**
  - A symbol called variable or non-terminals
  - A right arrow
  - **A string that contains other variables and terminals**

The Basics
Grammar Components

- **An example context free grammar:**
  - $S \rightarrow 0S1$
  - $S \rightarrow \epsilon$
- **Each production rule ($V \rightarrow w$) defines…**
  - How to replace a variable V with a string w regardless of the current context.
  - Do this repeatedly until there is no variable left.
  - The result is a string over all terminals.

The Basics
Grammar Components

- **An example context free grammar:**
  - $\boldsymbol{S} \to 0S1$
  - $S \to \epsilon$
- **One more thing: The start variable**
  - Defines the starting point of a sequence of substitutions.
  - Usually, it is the variable of the very first production rule.

The Basics
Formal Definition

- **Components: The 4-tuple $(\boldsymbol{V}, \boldsymbol{\Sigma}, \mathbf{R}, \boldsymbol{S})$**
  - V : A finite set of variables
  - Σ : A finite set of terminals that is disjoint from V
  - R : A finite set of production rules
  - S : The start variable $(S \in V)$

The Basics
Formal Definition - Example

- **Components: The 4-tuple** (V, Σ, R, S)
  - V : A finite set of variables
  - Σ : A finite set of terminals that is disjoint from V
  - R : A finite set of production rules
  - S : The start variable ($S \in V$)
- **An example grammar:** $V = \{S\}; \Sigma = \{0, 1, \epsilon\}; S_{start} = S$
  - $S \rightarrow 0S1$
  - $S \rightarrow \epsilon$

OR: $S \rightarrow 0S1 \mid \epsilon$

The empty string

The Basics
Derivation

- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - Step 2: Select a variable on the paper.
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.

The Basics
Derivation – Example 1

- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - Step 2: Select a variable on the paper.
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.
  - **Grammar:** $S \rightarrow 0S1 \mid \epsilon$
  - **Derivation:** S

The Basics
Derivation – Example 1

- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - Step 2: Select a variable on the paper.
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.
  - **Grammar:** $S \rightarrow 0S1 \mid \epsilon$
  - **Derivation:** $S \Rightarrow 0S1$

yields

The Basics
Derivation – Example 1

- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - Step 2: Select a variable on the paper.
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.
  - **Grammar:** $S \rightarrow 0S1 \mid \epsilon$
  - **Derivation:** $S \Rightarrow 0S1 \Rightarrow 00\boldsymbol{S}11 \Rightarrow 00\boldsymbol{\epsilon}11 = 0011$

The Basics
Derivation – Example 2

- **Generate a string from the start variable**
  - **Grammar:** $S \rightarrow aSa \mid bSb \mid \epsilon$
  - **Derivation:**

$$
\begin{aligned}
S &\Rightarrow aSa \\
&\Rightarrow aaSaa \\
&\Rightarrow aabSbaa \\
&\Rightarrow aab\epsilon baa \\
&= aabbaa
\end{aligned}
$$

  - $PALINDROME_{\{a,b\}}$

The Basics
Derivation

- **Context Free Grammar G =** $(\boldsymbol{V}, \boldsymbol{\Sigma}, \mathbf{R}, \boldsymbol{S})$
  - V : A finite set of variables
  - Σ : A finite set of terminals that is disjoint from V
  - R : A finite set of production rules
  - S : The start variable ($S \in V$)

derives

- **Context Free Language** $\boldsymbol{L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}}$
  - The set of all strings derived from the start variable.

The Basics
Derivation – Potential Problem?

- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - **Step 2: Select a variable on the paper.**
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.
- **What if there are multiple variables on the paper?**
  - Which one should be replaced next?

The Basics
Derivation – Leftmost versus Rightmost

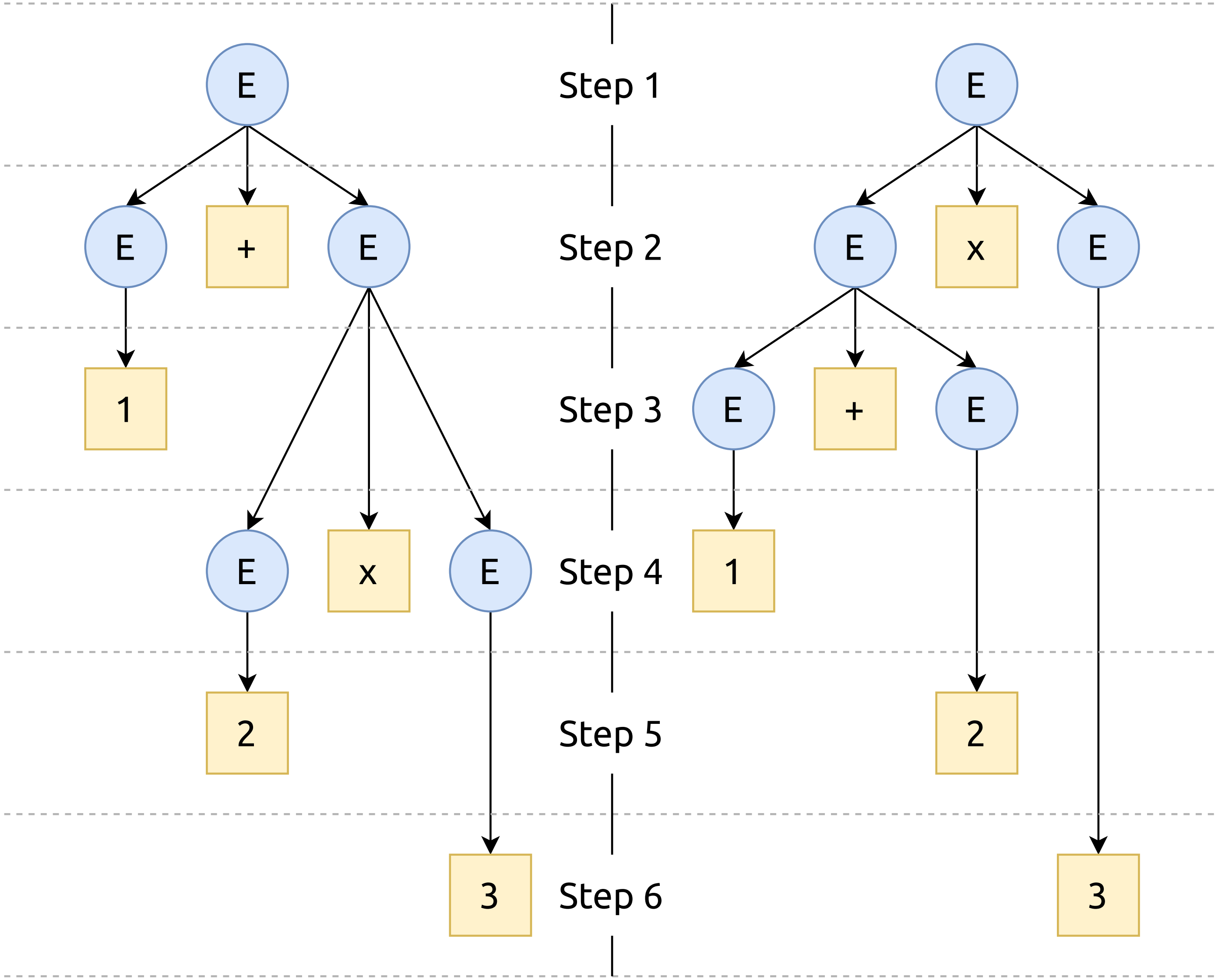
- **Generate a string from the start variable**
  - Step 1: Write down the start variable.
  - **Step 2: Select a variable on the paper.**
    - **Leftmost Derivation:** Always replace the leftmost variable in each step.
    - **Rightmost Derivation:** Always replace the rightmost variable in each step.
  - Step 3: Find the rule that has the selected variable on the left-hand side.
  - Step 4: Replace the selected variable with the right-hand side of that rule.
  - Step 5: Repeat Steps 2 – 5 until there is no variable left on the paper.

The Basics
Visualize Derivation

- **An example problematic grammar**
  - $E \to E + E \mid E \times E \mid n$
  - where E stands for *Expression* and n is any integer literal
- **Derive the string $1 + 2 \times 3$ from E**

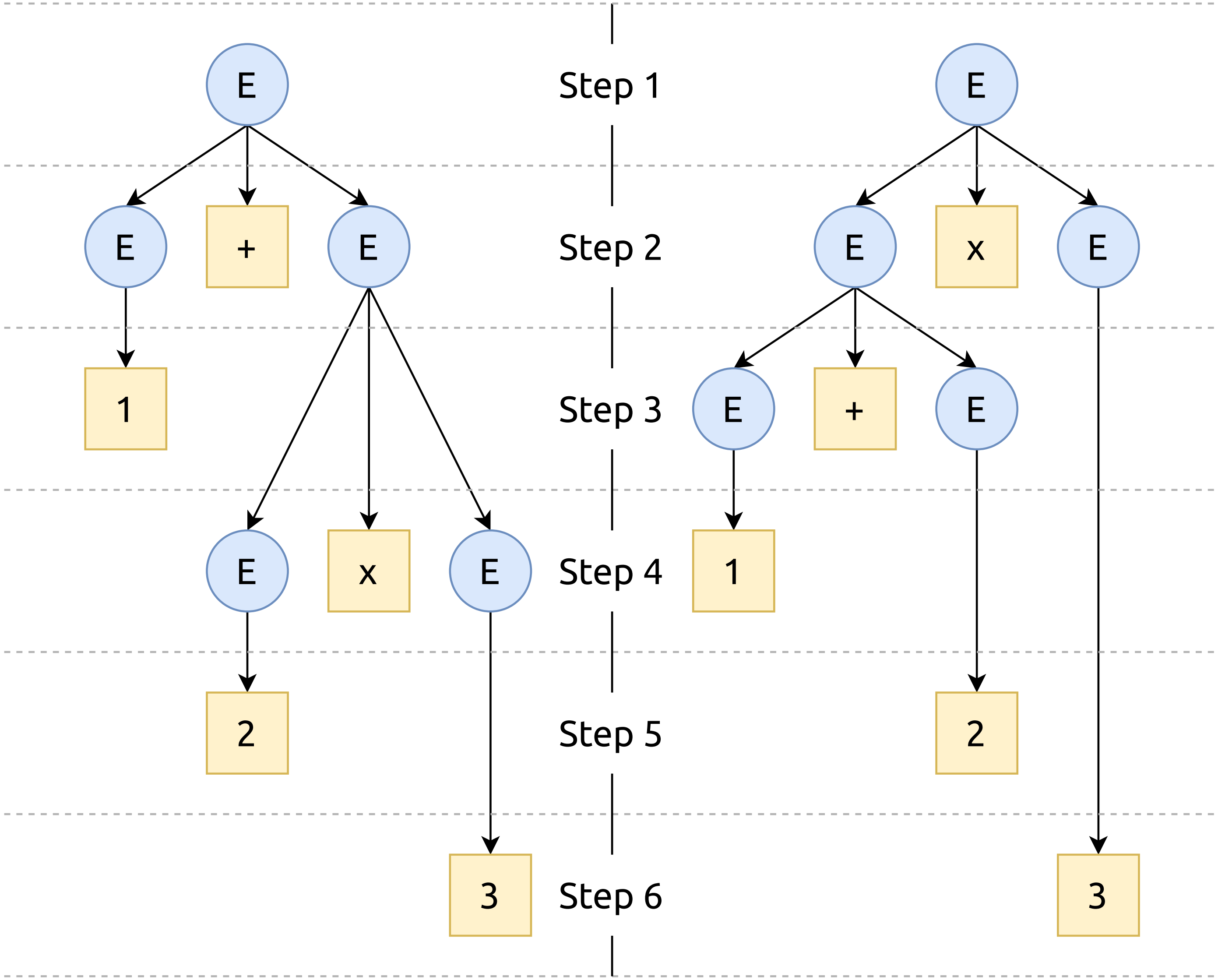
The Basics
Visualize Derivation – Leftmost

- $E \to E + E \mid E \times E \mid n$
- String: $1 + 2 \times 3$
- Two leftmost derivations
  - Also, two meanings ☹
  - $1 + (2 \times 3)$
  - $(1 + 2) \times 3$



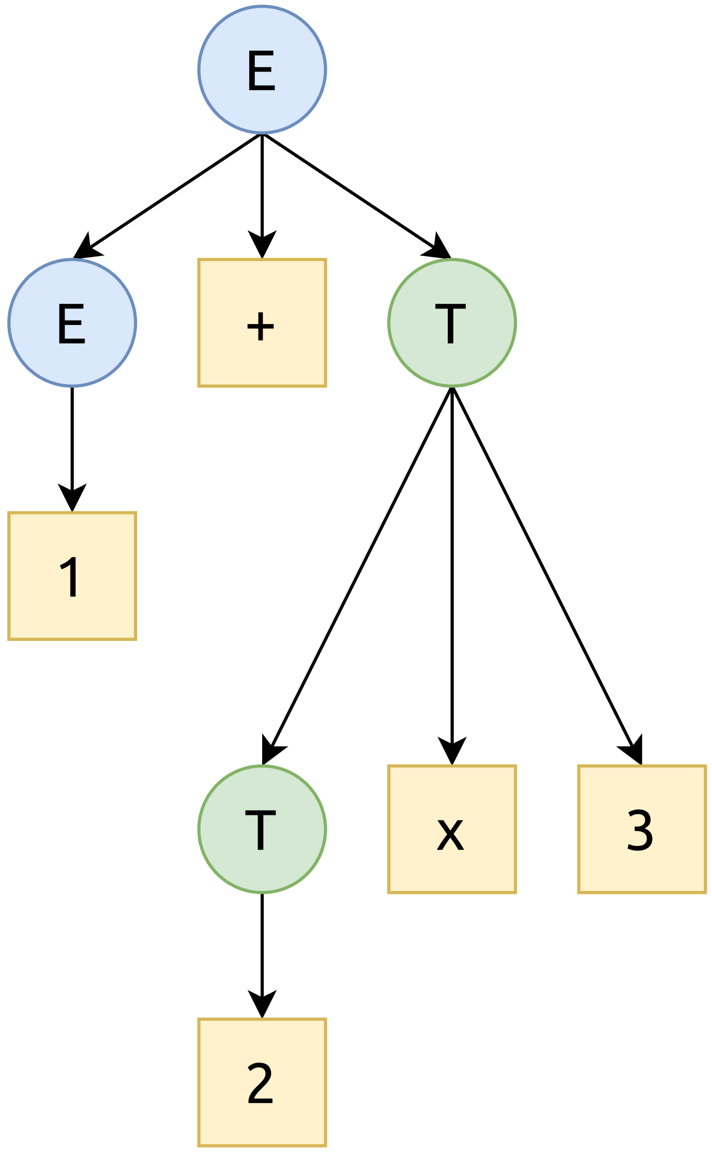
The Basics
Ambiguous Grammar

- A context free grammar is ambiguous if a derived string has more than one distinct leftmost derivation.
  - $1 + 2 \times 3 = 9 \; or \; 7?$
  - The compiler may evaluate the above expression to 9.



The Basics
Ambiguous Grammar

- **An example problematic grammar ☹**
  - $E \rightarrow E + E \mid E \times E \mid n$
- **Fixed grammar without ambiguity ☺**
  - $E \rightarrow E + T \mid T$
  - $\boldsymbol{T \rightarrow T \times n \mid n}$
  - $1 + 2 \times 3$ has only one leftmost derivation now.



Parsing Part 1
The Fundamental Idea

- **Pushdown Automaton (PDA)**
  - Finite State Automaton + A stack with unlimited amount of memory.
  - The machine can also push/pop a symbol onto/from the stack.
  - A set of input symbols + A set of stack symbols.
- **Recognize** $\mathbf{L = \{0^n 1^n \mid n \geq 0\}}$
  - Push “0” onto the stack when the machine reads a “0” from the tape.
  - Pop “0” from the stack when the machine reads a “1” from the tape.
  - Accept the input if the stack is empty on reading an “ϵ” from the tape.

Parsing Part 1
CYK Algorithm

- **Originally published by Itiroo Sakai in 1961.**
  - Sakai, Itiroo (1962). Syntax in universal translation.
  - 1961 International Conference on Machine Translation of Languages and Applied Language Analysis
- **But named after its rediscoverers:**
  - John Cocke
  - Danial Younger
  - Tadao Kasami

Parsing Part 1
CYK Algorithm

- **Exploit the idea of dynamic programming**
  - Use the solution to a smaller problem to solve a bigger problem.
- **The standard version has an important assumption.**
  - The grammar must be rendered into Chomsky Normal Form (CNF).
  - CNF defines constraints on each production rule.
- **There are variants that relax some of the constraints.**
  - "To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm" by Lange, Martin; Leiß, Hans in 2009.

Parsing Part 1
Chomsky Normal Form (CNF)

- **Every production rule must be of the form**
  - $A \to BC$
  - OR
  - $A \to a$
- **Notes**
  - A, B, C are any variables, and a is any terminal.
  - B, C must not be the start variable.
  - $S \to \epsilon$ is allowed, if S is the start variable.

Parsing Part 1
Chomsky Normal Form (CNF)

- **Every production rule must be of the form**
  - $A \to BC$
  - OR
  - $A \to a$
- **Observations**
  - A variable can be directly replaced by a terminal.
  - Otherwise, a variable is separated into two parts.
    - Each part is replaced by some other string.

Parsing Part 1
Chomsky Normal Form (CNF)

- **[Sip] Every context free grammar can be transformed into CNF.**
- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - BIN: Eliminate right-hand sides with more than 2 variables.
  - DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.
  - UNIT: Eliminate all unit rules ($A \to B$).

Parsing Part 1
Chomsky Normal Form (CNF)

$\mathbf{S} \rightarrow a\mathbf{S}a \mid b\mathbf{S}b \mid \epsilon$

$\mathbf{S'} \rightarrow \mathbf{S}$

$S \rightarrow aSa \mid bSb \mid \epsilon$

- **The transformation is done in 5 steps:**
  - **START: Eliminate the start variable from the right-hand sides.**
    - Introduce a new start variable S' that derives the original start variable S.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - BIN: Eliminate right-hand sides with more than 2 variables.
  - DEL: Eliminate all ϵ-rules ($A \rightarrow \epsilon$) not involving the start variable.
  - UNIT: Eliminate all unit rules ($A \rightarrow B$).

Parsing Part 1
Chomsky Normal Form (CNF)

$S' \to S$
$S \to aSa \mid bSb \mid \epsilon$

$S' \to S$
$S \to ASA \mid BSB \mid \epsilon$
$A \to a$
$B \to b$

- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - **TERM: Eliminate right-hand sides with both variables and terminals.**
    - Introduce a new variable X_i for each terminal x_i on the right-hand side.
    - Introduce a new production rule $X_i \to x_i$.
  - BIN: Eliminate right-hand sides with more than 2 variables.
  - DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.
  - UNIT: Eliminate all unit rules ($A \to B$).

Parsing Part 1
Chomsky Normal Form (CNF)

$S' \to S$
$S \to ASA \mid BSB \mid \epsilon$
$A \to a; B \to b$

$S' \to S$
$S \to AX \mid BY \mid \epsilon$
$X \to SA; Y \to SB$
$A \to a; B \to b$

- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - **BIN: Eliminate right-hand sides with more than 2 variables.**
    - $A \to X_1X_2 \dots X_n; Let\ Head = X_1; Let\ Tail = X_2X_3 \dots X_n$:
    - Recursively replace the tail sequence of variables with a new variable until $|Tail| = 2$.
  - DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.
  - UNIT: Eliminate all unit rules ($A \to B$).

Parsing Part 1
Chomsky Normal Form (CNF)

$S' \to \mathbf{S}$
$S \to AX \mid BY \mid \epsilon$
$X \to \mathbf{S}A; Y \to \mathbf{S}B$
$A \to a; B \to b$

$S' \to S \mid \epsilon$
$S \to AX \mid BY$
$X \to SA \mid \mathbf{A}; Y \to SB \mid \mathbf{B}$
$A \to a; B \to b$

- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - BIN: Eliminate right-hand sides with more than 2 variables.
  - **DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.**
    - For each occurrence of an A on the right-hand side:
      - Add a new rule with that occurrence deleted.
  - UNIT: Eliminate all unit rules ($A \to B$).

Parsing Part 1
Chomsky Normal Form (CNF)

$S' \to S \mid \epsilon$
$S \to AX \mid BY$
$X \to SA \mid A; Y \to SB \mid B$
$A \to a; B \to b$

$S' \to AX \mid BY \mid \epsilon$
$S \to AX \mid BY$
$X \to SA \mid a; Y \to SB \mid b$
$A \to a; B \to b$

- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - BIN: Eliminate right-hand sides with more than 2 variables.
  - DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.
  - **UNIT: Eliminate all unit rules ($A \to B$).**
    - Whenever $B \to v$ appears, add a rule $A \to v$.

Parsing Part 1
Chomsky Normal Form (CNF)

- **The transformation is done in 5 steps:**
  - START: Eliminate the start variable from the right-hand sides.
  - TERM: Eliminate right-hand sides with both variables and terminals.
  - **BIN: Eliminate right-hand sides with more than 2 variables.**
  - **DEL: Eliminate all ϵ-rules ($A \to \epsilon$) not involving the start variable.**
  - UNIT: Eliminate all unit rules ($A \to B$).
  - More details and time analysis are covered in the textbook and the paper.
    - "To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm"

Order Matters

Parsing Part 1
CYK Algorithm

- Given a CFG G in CNF and an input string w of length n.
- **Exploit the properties of CNF:** $A \rightarrow BC$ or $A \rightarrow a$; $S \rightarrow \epsilon$ is allowed.
  - Supposed that the input string can be generated from G…
  - If a string w is ϵ, then there exists a rule $S \rightarrow \epsilon$.
  - If a string w of length 1 can be derived from a variable A,
    - then there exists a rule $A \rightarrow w$.
  - If a string w of length ≥ 2 can be derived from a variable A…
    - then there exists a rule $A \rightarrow BC$ such that
      - B derives the substring w_{front} (← A smaller problem)
      - C derives the substring w_{back} (← A smaller problem)
      - where $w = w_{front} + w_{back}$ (string concatenation)

Parsing Part 1
CYK Algorithm

- Exploit the properties of CNF: $A \to BC$ or $A \to a$; $S \to \epsilon$ is allowed.
  - If a string w of length ≥ 2 can be derived from a variable A...
    - **Then there exists a rule $A \to BC$ such that**
      - **B derives the substring w_{front} (← A smaller problem)**
      - **C derives the substring w_{back} (← A smaller problem)**
      - where $w = w_{front} + w_{back}$ (string concatenation)
    - **Where should we split w into w_{front} and w_{back}?**
      - We need to try every possible partitions.
    - Good! We reduce a big problem into two smaller problems!
      - Top-down Approach: We could recursively solve the problem now.

Parsing Part 1
CYK Algorithm

- **Bottom-up Approach:**
  - If we know which variables generate all substrings of the input up to length k, can we know which variable generates a particular substring of length $k+1$?
    YES!
    - Split a substring of length $k+1$ into two non-empty pieces (there are k possible ways).
    - For each rule of form $A \rightarrow BC$:
      - Check whether B can generate the first piece of length $p \leq k$.
      - Check whether C can generate the second piece of length $k+1-p \leq k$.
      - If so, then A can generate this substring of length $k+1$.
    - Now we just check every possible substring of length $k+1$.

Parsing Part 1
CYK Algorithm

- **Bottom-up Approach:**
  - If we know which variables generate all substrings of the input up to length k, we know which variable generates a particular substring of length $k + 1$?
  - By induction, we know which variables generate the substring of length n.
    - Substring of length n is just the input string.
    - If those variables contain the start variable S, then $w \in L(G)$.

Parsing Part 1
CYK Algorithm

- $Input$ = $< G_{CNF} = (V, \Sigma, R, S), w = \sigma_1\sigma_2 \dots \sigma_n >$; $Output$ = $accept$ or $reject$.
- $Table$ = $n \times n$ cells
  - where $Table[i, j]$ stores a set of variables that can generate the substring $\sigma_i\sigma_{i+1} \dots \sigma_j$ $(i \leq j)$.
- If w is empty, if $S \rightarrow \epsilon$ exists then $accept$ else $reject$.
- For $i = 1 \dots n$:
  - For each variable A: If $A \rightarrow \sigma_i$ exists, then insert A into $Table[i, i]$.
- For $l = 2 \dots n$:
  - For $i = 1 \dots (N - l + 1)$:
    - Let $j = i + l - 1$; For $k = i \dots (j - 1)$:
      - For each rule $A \rightarrow BC$: If $Table[i, k]$ contains B and $Table[k + 1, j]$ contains C, then insert A into $Table[i, j]$.
- If $Table[1, n]$ contains S then accept else reject.

Parsing Part 2
Practical Parsers

- The standard CYK algorithm only tells us whether an input string can be generated.
- Sometimes, we also want to know ***how a string is generated***.
  - e.g., A compiler needs to convert the source code to an abstract syntax tree so that it can perform type checking and produce the assembly code.
  - i.e., Search for the derivation from S to the input string w.

Parsing Part 2
Parser Types

- **Top-down Parsers**
  - Build a derivation *from the start variable to the input string*.
  - At each step, the parser selects a variable A and replaces the variable with the right-hand side of the rule $A \to v$.
- **Bottom-up Parsers**
  - Build a derivation *from the input string back to the start variable*.
  - At each step, the parser identifies a substring v that matches the right-hand side of a rule $A \to v$ and replaces the substring with the variable.

Parsing Part 2
Top-down Parsers

- **Begin with the start variable…**
  - At each step, the parser selects a variable and replaces the variable with the right-hand side of the rule.
  - Keep expanding the parse tree until the leaves match the input string.
- **Example with input string $\boldsymbol{bacab}$:**
  - Derivation: $S \Rightarrow d_1 \Rightarrow d_2 \Rightarrow \cdots \Rightarrow d_{n-1} \Rightarrow d_n = bacab$
  - Grammar: $S \to b\ A\ C\ b;\ A \to aA \mid c;\ C \to cC \mid a$
  - $\boldsymbol{d_i = baACb}$**, so** $\boldsymbol{d_{i+1}}$ **can be one of:**
    - $baaACb\ (A \to aA)$
    - $bacCb\ (A \to c)$
    - $baAcCb\ (C \to cC)$
    - $baAab\ (C \to a)$

Parsing Part 2
Parser Types

- **Top-down Parsers**
  - Recursive descent parsers (with backtracking)
  - Predictive parsers: $LL(k)$ parsers (without backtracking)
    - Read the input **L**eft to right; Build **L**eftmost derivation; Peek at most k symbols.
- **Bottom-up Parsers**
  - Shift-reduce parsers (without backtracking)
  - $LR(k)$ parsers (without backtracking)
    - Read the input **L**eft to right; Build **R**ightmost derivation in reverse; Peek at most k symbols.

Parsing Part 2
$LL(1)$ Parser – A Quick Glance

- **Peek the next symbol is sufficient to choose the correct production rule**
  - $S \rightarrow aP \mid bQ$
  - Supposed that the parser is parsing the variable S.
    - If the next symbol is a, the parser consumes a and starts to parse the variable P.
    - If the next symbol is b, the parser consumes b and starts to parse the variable Q.
- **Constraints on the context free grammar**
  - The constrained grammar is known as $LL(1)$ grammar.
  - The first symbol of all strings derived from a variable must be unique.
    - $S \rightarrow aP \mid bQ \mid aR$

Parsing Part 2
$LL(1)$ Parser – Constraints

- **Constraints on the context free grammar**
  - The constrained grammar is known as $LL(1)$ grammar.
  - The **first symbol** of all strings derived from a variable must be unique.
    - Problematic ☹:
      - $S \rightarrow \boldsymbol{a}P \mid bQ \mid \boldsymbol{a}R$
    - Fixed ☺:
      - $S \rightarrow \boldsymbol{aX} \mid bQ$
      - $\boldsymbol{X \rightarrow Q \mid R}$
      - $Q \rightarrow c \mid q$
      - $R \rightarrow d \mid r$

Parsing Part 2
$LL(1)$ Parser – Constraints

- **Constraints on the context free grammar**
  - The constrained grammar is known as $LL(1)$ grammar.
  - The first symbol of all strings derived from a variable must be unique.
  - Left recursion is not allowed.
    - $E \rightarrow E + T \mid T$
    - $T \rightarrow T \times n \mid n$
    - When the parser is parsing E...
      - It needs to parse E, then $+$, and finally T.
        - It needs to parse E, ...
          - Stack overflow.

Parsing Part 2
$LL(1)$ Parser – Constraints

- **Constraints on the context free grammar**
  - The constrained grammar is known as $LL(1)$ grammar.
  - The first symbol of all strings derived from a variable must be unique.
  - Left recursion is not allowed.
    - $E \rightarrow E + T \mid T$
    - $T \rightarrow T \times n \mid n$
  - Left recursions removed:
    - $E \rightarrow TZ; Z \rightarrow +E \mid \epsilon$
    - $T \rightarrow nR; R \rightarrow \times T \mid \epsilon$

Parsing Part 2

$LL(1)$ Parser – JavaCC Example

- **Generate a parser for the grammar:**
  - $E \rightarrow T + E \mid T - E \mid T$
  - $T \rightarrow n \times T \mid n$

  - $E \rightarrow T\left((+|-)\ T\right) *$
  - $T \rightarrow P\ (\times\ P) *$
  - $P \rightarrow n$

Context Free Grammar
Summary

- **The Basics**
  - Syntax: $A \rightarrow w$
  - Formal Definition: $G = (V, \Sigma, R, S)$
  - Derivation: $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n; S \Rightarrow^* w_n$
    - Leftmost Derivation versus Rightmost Derivation
    - Ambiguous Grammar
    - Parse Tree: Visual Derivations

Context Free Grammar
Summary

- **Parsing**
  - Pushdown Automaton: Finite State Automaton + Stack
  - Chomsky Normal Form: Constraints and Transformations
  - Cocke-Younger-Kasami Algorithm (CYK Algorithm)
  - Top-Down Parsing versus Bottom-Up Parsing
    - Recursive Descent Parsers
      - $LL(k)$ Parsers
        - $LL(1)$ Parsers: Constraints and Solutions

Context Free Grammar
Questions?

Thanks for joining today

☺

Any Questions?

Context Free Grammar
References & Notes

- **"Three models for the description of language"**
  - Noam Chomsky (1956), IRE Transactions on Information Theory.
- **"To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm"**
  - Martin Lange, Hans Leiß (2009), Informatica Didactica.
- **"Introduction to the Theory of Computation"**
  - Section 2.1 and Section 2.2: Basics of CFG and PDA.
  - Chapter 7: Section 7.2 Theorem 7.16: The CYK Algorithm.

Context Free Grammar
References & Notes

- **“Modern Compiler Implementation in Java”**
  - Second Edition, Andrew Appel, 2002
- **“Comparison of parser generators – Deterministic CFG"**
  - https://en.wikipedia.org/wiki/Comparison_of_parser_generators#Deterministic_context-free_languages
- **“JavaCC Parser Generator”**
  - https://javacc.github.io/javacc/