Chapter 5: Context-Free Languages

Dr. Yuan Yao

University of Nottingham Ningbo China (UNNC)

Learning Outcomes

Learning outcomes

At the conclusion of this chapter, the students are expected to be able to:

- Identify whether a particular grammar is context-free.
- Discuss the relationship between regular languages and context-free languages.
- Construct context-free grammars for simple languages.
- Produce leftmost and rightmost derivations of a string generated by a context-free grammar.
- Construct derivation trees for strings generated by a context-free grammar.
- Learn what it means for a context-free grammar to be ambiguous.
- For a given ambiguous grammar, find an equivalent grammar which is not ambiguous, if possible.

Introduction

Introduction

- As we have discussed before, the theory of formal languages is vital for:
  - Definition of programming languages.
  - Constructions of translators, i.e., interpreters and compilers.

- Translation programs rely on a given specification of a language.
- As such, the specification of a programming language must be *precise*.
- A good specification should provide the foundation for the writing of **reliable** translation programs.
- As we have seen, regular languages are used in the recognition of certain simple patterns (e.g., identifiers and literals in C).

- Can we directly use *regular languages* to complete these tasks?

Introduction

- The **regular language** can be expanded to a language called **context-free language**.
    - $L = \{ a^n b^n \mid n \geq 0 \}$ is an example.

- We have three different ways to represent regular languages:
    - FA (DFA, NFA)
    - Regular Expressions
    - Regular Grammars

- These representation can also be extended to context-free languages, except the **regular expression**.

Introduction

- The **regular language** can be expanded to a language called **context-free language**.
  - $L = \{ a^n b^n \mid n \geq 0 \}$ is an example.
- We have three different ways to represent regular languages:
  - FA (DFA, NFA) $\Rightarrow$ Pushdown automata
  - Regular Expressions
  - Regular Grammars $\Rightarrow$ Context-free Grammar
- These representation can also be extended to context-free languages, except the **regular expression**.

Context-Free Grammars

Context-Free Grammars

- Definition: We call $G = (V, T, S, P)$ a **context-free grammar (CFG)** if all the productions in P have the form:

$$A \to x$$

in which $A \in V$, and $x \in (V \cup T)^*$.

- The left-hand side of each production is a single variable, where there is no restrictions on the right-hand side.
- We say that L is a **context-free language (CFL)** if and only if there is a context-free grammar G such that $L = L(G)$, that is, L is generated by G.

Context-Free Grammars

What's the differences the **context-free grammars** and the **regular grammars**?

- **Regular Grammars:** There are restrictions on the productions allowed for regular grammar:
  - Left-linear grammar.
  - Right-linear grammar, e.g., $S \to abS|\lambda$, and it generates a regular language $L((ab)^*)$.

- **Context-Free Grammars:** No restrictions on the right-hand side of the productions.
  - Therefore, the class of CFLs includes the class of regular languages as a proper subset.

Context-Free Grammars

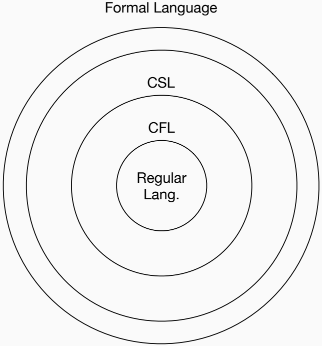
- Note that, a context-free grammar does impose a restriction on the left side of productions:
    - **i.e, the left-hand side must be a single variable**.
- For instance, a production of the form:

$$abSc \to abSSc$$

  is not allowed, because the left side $abSc$ is not a single variable.
- However, context-free languages is not equivalent to the formal language, i.e., there are some formal languages which do not belong to the context-free languages.

Context-Sensitive Grammars

- If we relax the constraints on the left-hand side of a production in a context-free grammar, then we have **context-sensitive grammars (CSG)**:
  - If $\alpha \to \beta$, then we require $|\alpha| \leq |\beta|$.
- For example, $abSc \to abac$ is allowed in a context-sensitive grammar.



CFG Versus CSG

- What's the difference between **Context-free Grammars** and **Context-sensitive Grammars**?
- Why they are called context-free and context-sensitive?
- Consider a grammar $G = (\{S\}, \{a, b, c\}, S, P)$, where P is defined as:

$$
\begin{aligned}
S &\rightarrow aSa \\
S &\rightarrow bSbb \\
abSc &\rightarrow abcc
\end{aligned}
$$

Example: Context-Free Languages

- Consider a grammar $G = (\{S\}, \{a, b\}, S, P)$, where P is defined as:

$$S \rightarrow \quad aSb \quad | \quad SS \quad | \quad \lambda$$

- Write down some sample derivations and find out what is the language generated by the above grammar?

Derivations

Derivations

- One significant difference between context-free grammar and regular grammar is that there is no restrictions on the right-hand side of a production.
- Therefore, we could have multiple variables on the right side.
- Given a CFG, $G = (\{S, A, B\}, \{a, b\}, S, P)$, where P is defined as follows:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aaA \,|\, \lambda \\
B &\rightarrow Bb \,|\, \lambda
\end{aligned}
$$

- What is the language?
- Exercise: Write down a derivation process for the string aab

Leftmost and Rightmost Derivation

- In a **leftmost derivation (LMD)**, at each step, the leftmost variable in a sentential form is replaced.
- In a **rightmost derivation (RMD)**, at each step, the rightmost variable in a sentential form is replaced.
- Given a CFG, $G = (\{S, A, B\}, \{a, b\}, S, P)$, where P is defined as follows:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aaA \mid \lambda \\ B &\rightarrow Bb \mid \lambda \end{aligned}$$

- Exercise: Write the LMD and RMD for aab.

Leftmost and Rightmost Derivation

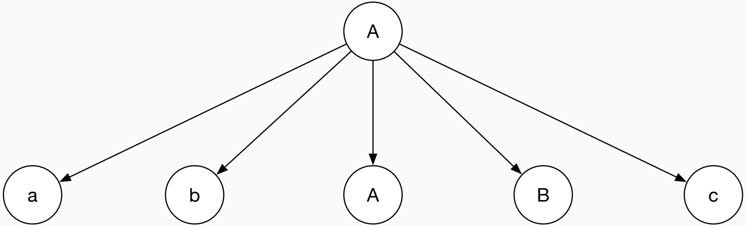
- In a **leftmost derivation (LMD)**, at each step, the leftmost variable in a sentential form is replaced.
- In a **rightmost derivation (RMD)**, at each step, the rightmost variable in a sentential form is replaced.
- Given a CFG, $G = (\{S, A, B\}, \{a, b\}, S, P)$, where P is defined as follows:

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow aaA \,|\, \lambda \\
B &\rightarrow Bb \,|\, \lambda
\end{aligned}
$$

- Exercise: Write the LMD and RMD for aab.
  - **LMD:** $S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$
  - **RMD:** $S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$

Derivation Tree

- In a *derivation tree* or *parser tree*,
  - The root is labeled with the start variable.
  - Internal (non-leaf) nodes are labelled with a variable occurring on the left side of a production.
  - The children of a node contain the symbols on the corresponding right side of a production.
- Given the production $A \to abABc$, a partial derivation tree can be draw as follows:



Derivation Tree

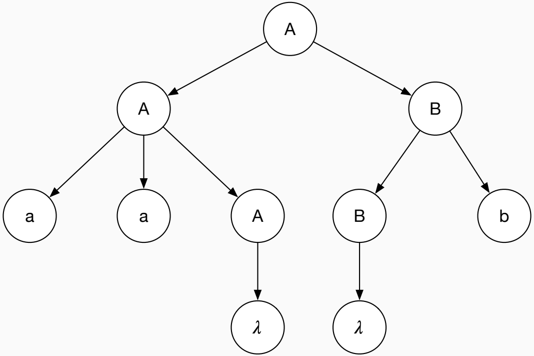
- Let $G = (V, T, S, P)$ be a context-free grammar. An ordered tree is a derivation tree for G if and only if it has the following properties.
  - The root is labeled S.
  - Every leaf has a label from $T \cup \{\lambda\}$.
  - Every interior node (non-leaf node) has a label from V.
  - If a node has label $A \in V$, and its children are labeled (from left to right) $a_1, a_2, \ldots, a_n$, then P must contain the following production:

$$A \to a_1 a_2 \ldots a_n$$

  - A leaf labeled λ has no siblings; that is, a node with a child labeled λ can have no other children.

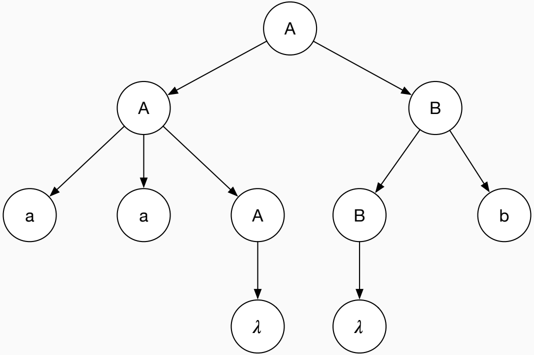
Derivation Tree

- The string of symbols obtained by reading the leaves of the tree from left to right, omitting any λ's, is said to be the **yield** of the tree.
- The **yield** of a derivation tree is the string of terminals produced by a leftmost depth-first traversal of the tree.



Derivation Tree

- Question: What is the yield of this derivation tree?



Derivations and Derivation Trees

- What's the relationship between derivations and derivation trees?
    - Derivation trees for derivations are like transition graphs for finite automata.

- **Theorem 5.1** Let $G = (V, T, S, P)$ be a CFG. Then for every $w \in L(G)$, there exists a derivation tree of G whose yield is w. Conversely, the yield of any derivation tree is in $L(G)$. Also, if t_G is any partial derivation tree for G whose root is labeled S, then the yield of t_G is a sentential form of G.

- Note: Derivation trees show which productions are used in obtaining a sentence, but do not give the order of their application.

Parsing and Ambiguity

Parsing and Membership

- **The parsing problem:** Given a grammar G and a string w, find a sequence of derivations using the productions in G to produce w.
- The parsing problem is a central problem in compilers.

- **The membership algorithm:** an algorithm that can tell us whether w is in $L(G)$ or not.

- **How to solve the membership problem?** Regardless of efficiency.

Exhaustive parsing

- **How to solve the membership problem?** Regardless of efficiency.
  - Exhaustive parsing.
- **Exhaustive parsing:** Systematically construct all possible (e.g., leftmost) derivations and see whether any of them match the given string w.
- This can be easily done by using the following procedure:
  - Looking at all productions of the form $S \to x$.
  - Finding all x that can be derived from S in one step.
  - If none of these results in a match with w, we apply all applicable productions to the left most variable of every x.
  - ...

Exercise: Exhaustive parsing

- Consider the following grammar G:

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

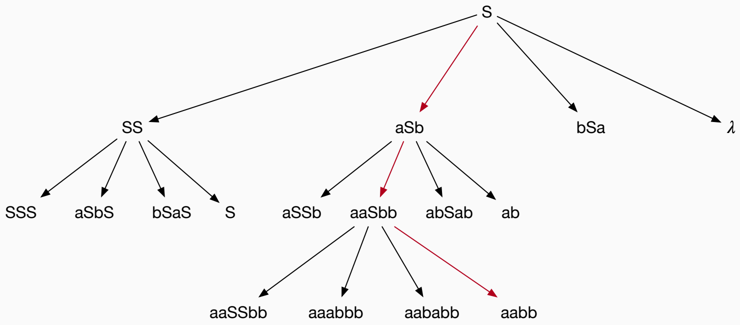
  and the string $w = aabb$. Use exhaustive parsing to check whether w is in $L(G)$ or not.

Exercise: Exhaustive parsing

- Consider the following grammar G:

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

and the string $w = aabb$. Use exhaustive parsing to check whether w is in $L(G)$ or not.



Problems of the Exhaustive parsing

- Consider the following grammar G:

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

  and the string $w = aabb$. Use exhaustive parsing to check whether w is in $L(G)$ or not.

- Given the string abb to check, what will happen?

Problems of the Exhaustive parsing

- Consider the following grammar G:

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

  and the string $w = aabb$. Use exhaustive parsing to check whether w is in $L(G)$ or not.

- Given the string abb to check, what will happen?
  - Tediousness.
  - Not terminate for strings not in $L(G)$
- How to solve it?

Exhaustive parsing

- The nontermination problem of the exhaustive parsing can be solved by restricting the form of the CFG as follows:
  - No λ-productions, e.g., $A \to \lambda$ are allowed.
  - and no unit-productions, e.g., $A \to B$ are allowed.
- We could deduce that no derivations of a non-empty string x can take more than $2|x| - 1$ steps.
- How to prove it?

- **Theorem 5.2:** Given a context-free grammar $G = (V, T, S, P)$ that does not have any productions of the following forms:

$$A \to \lambda$$

or

$$A \to B$$

where $A, B \in V$. Then the exhaustive search parsing method can be made into an algorithm that, for any $w \in \Sigma^*$, either produces a parsing of w or tells us that no parsing is possible.

Example: Context-free Grammar

- Consider the following CFG G for generating simple algebraic expressions:

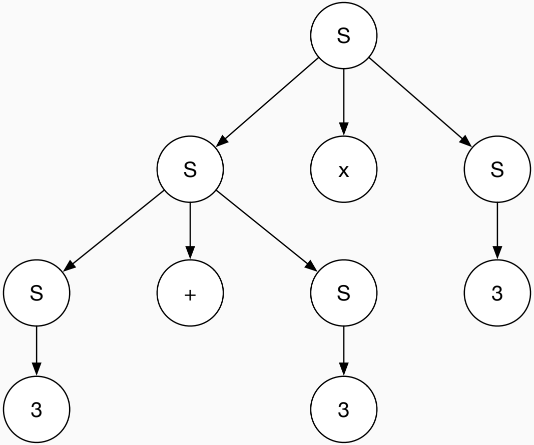
$$S \to S + S \mid S \times S \mid (S) \mid 3$$

- Find a derivation for the string $3 + 3 \times 3$.
- What is the result of $3 + 3 \times 3$?

Example: Context-free Grammar

- The derivation tree for the first derivation:

$$S \Rightarrow S \times S \Rightarrow S + S \times S \Rightarrow 3 + S \times S \Rightarrow 3 + 3 \times S \Rightarrow 3 + 3 \times 3$$



Example: Context-free Grammar

- Draw the derivation tree for the second derivation:

$$S \Rightarrow S + S \Rightarrow 3 + S \Rightarrow 3 + S \times S \Rightarrow 3 + 3 \times S \Rightarrow 3 + 3 \times 3$$

Parsing and Ambiguity

- A context-free grammar G is said to be **ambiguous** if there exists some $w \in L(G)$ that has at least two distinct derivation trees.
- Given a context-free grammar G, for any $x \in L(G)$, the following statements are equivalent:
  - x has more than one derivation tree.
  - x has more than one LMD.
  - x has more than one RMD.
- Why the concept of ambiguity is important?

Example: if-statement

- In C, an *if*-statement can be defined as:

$$
\begin{aligned}
S &\rightarrow \quad \mathit{if}\,(C) \quad S \\
&\rightarrow \quad \mathit{if}\,(C) \quad S \quad \mathit{else} \quad S \\
&\rightarrow \quad OS
\end{aligned}
$$

- Consider the following statement in C:

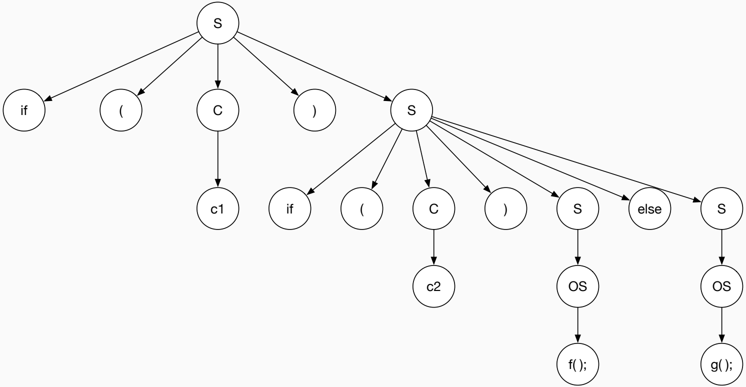
  if (c_1) if (c_2) f(); else g();

- Draw the derivation tree(s) for this statement.

Example: if-statement

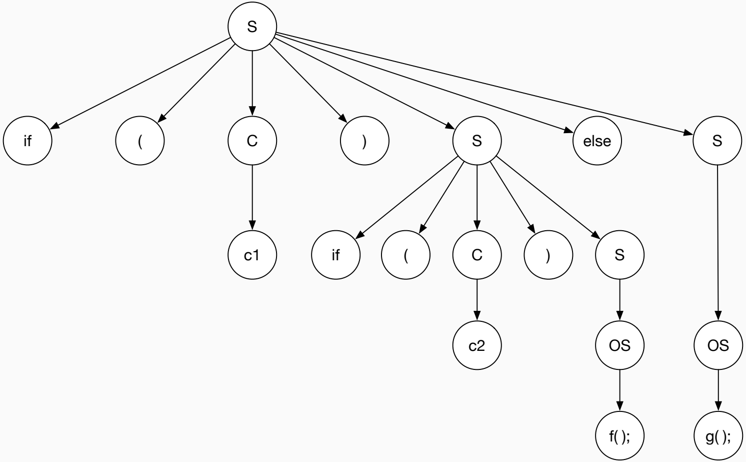
- Consider the following statement in C:

  if (c_1) if (c_2) f(); else g();



Example: if-statement

- Another possible derivation tree



Example: if-statement

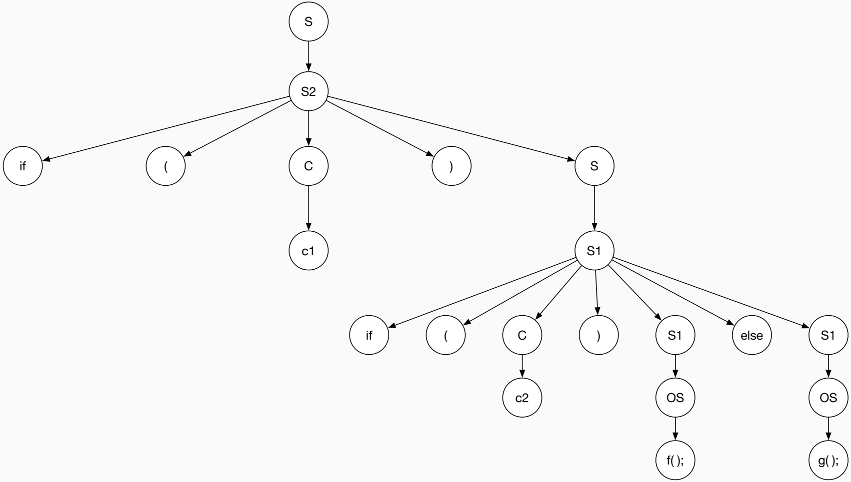
- It is possible to avoid such ambiguity by using the following grammar:

$$
\begin{array}{lcll}
S & \rightarrow & S_1 & \mid S_2 \\
S_1 & \rightarrow & \mathit{if}\ (C)\ S_1\ \mathit{else}\ S_1 & \mid OS \\
S_2 & \rightarrow & \mathit{if}\ (C)\ S & \mid \mathit{if}\ (C)\ S_1\ \mathit{else}\ S_2
\end{array}
$$

- The variable S_1 represents a statement in which every if is matched by a corresponding else, while every statement derived from S_2 contains at least one unmatched if.
- The only variable appearing before else in these rules is S_1; because the else cannot match any of the ifs in the statement derived from S_1, it must match the if that appeared at the same time as itself. Hence, the grammar is unambiguous.
- Try if (c_1) if (c_2) f(); else g();

Example: if-statement

- if (c_1) if (c_2) f(); else g();



Exercise

- Rewrite the following CFG G for generating simple algebraic expressions, such that the new grammar G' is not ambiguous.

$$S \to S + S \mid S \times S \mid 3$$

Ambiguous Languages

- For some languages, it is possible to find an unambiguous grammar, as shown in the previous examples.
- There are, however, **inherently** ambiguous languages, for which every possible grammar is ambiguous.
- Consider the language $\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$, which is generated by the following grammar:

$$
\begin{array}{lcl}
S & \rightarrow & S_1 \mid S_2 \\
S_1 & \rightarrow & S_1 c \mid A \\
A & \rightarrow & aAb \mid \lambda \\
S_2 & \rightarrow & aS_2 \mid B \\
B & \rightarrow & bBc \mid \lambda
\end{array}
$$

- Consider $a^n b^n c^n$