Lecture 13 – Parse Trees and Ambiguity

COSE215: Theory of Computation

Jihyeok Park

PLRG

2023 Spring

Recall

- A **context-free grammar (CFG)**:

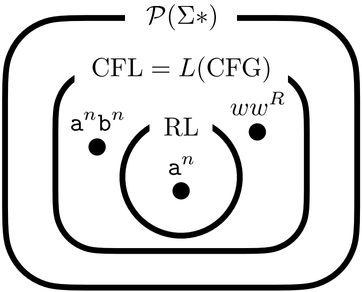
$$G = (V, \Sigma, S, P)$$

- The **language** of a CFG G:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

- A language L is a **context-free language (CFL)**:

$$\exists \text{ CFG } G.\ L(G) = L$$



Contents

Parse Trees

Consider the following CFG for arithmetic expressions:

$$
\begin{aligned}
S &\rightarrow N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

A derivation and parse tree for a sentential form $N\texttt{*}X\texttt{+}N$:

$$
\begin{aligned}
S &\Rightarrow S\texttt{+}S \\
&\Rightarrow S\texttt{*}S\texttt{+}S \\
&\Rightarrow N\texttt{*}S\texttt{+}S \\
&\Rightarrow N\texttt{*}X\texttt{+}S \\
&\Rightarrow N\texttt{*}X\texttt{+}N
\end{aligned}
$$

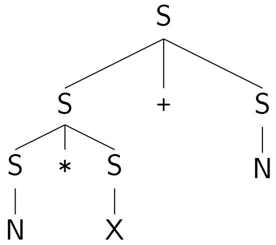
Parse Trees

Consider the following CFG for arithmetic expressions:

$$
\begin{array}{rcl}
S & \rightarrow & N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

A derivation and parse tree for a sentential form $N\texttt{*}X\texttt{+}N$:

$$
\begin{array}{rcl}
S & \Rightarrow & S\texttt{+}S \\
 & \Rightarrow & S\texttt{*}S\texttt{+}S \\
 & \Rightarrow & N\texttt{*}S\texttt{+}S \\
 & \Rightarrow & N\texttt{*}X\texttt{+}S \\
 & \Rightarrow & N\texttt{*}X\texttt{+}N
\end{array}
$$



Parse Trees

Definition (Parse Trees)

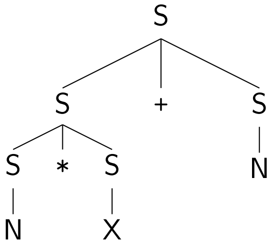
For a given CFG $G = (V, \Sigma, S, P)$, **parse trees** are trees satisfying:

1. The **root node** is labeled with the **start variable** S.
2. Each **internal node** is labeled with a **variable** $A \in V$.
   If its children are labeled with:
   $$X_1, X_2, \cdots, X_k$$
   from the left to the right, then $A \to X_1 X_2 \cdots X_k \in P$.
3. Each **leaf node** is labeled with a variable, symbol, or ϵ. However, if a leaf node is labeled with ϵ, it must be the only child of its parent.

Parse Trees – Example 1: Arithmetic Expressions

$$
\begin{aligned}
S &\rightarrow N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

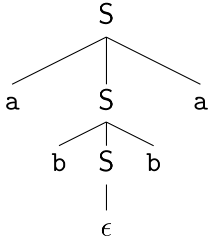
A parse tree:



Parse Trees – Example 2: Even Palindromes

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

A parse tree:



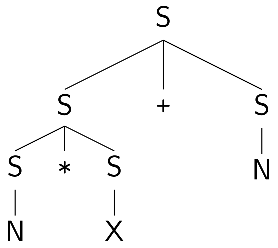
Yields

Definition (Yields)

The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree is called the **yield** of the parse tree.

Definition (Yields)

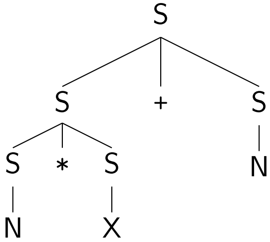
The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree is called the **yield** of the parse tree.



Yields

Definition (Yields)

The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree is called the **yield** of the parse tree.

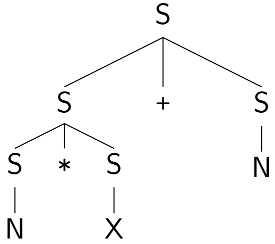


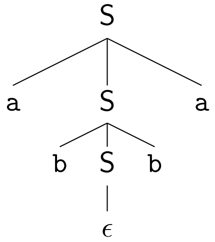
Its yield is $N{*}X{+}N$.

Yields

Definition (Yields)

The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree is called the **yield** of the parse tree.



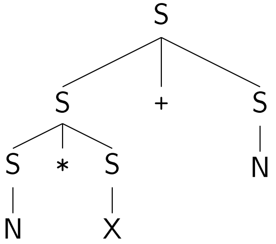


Its yield is $N{*}X{+}N$.

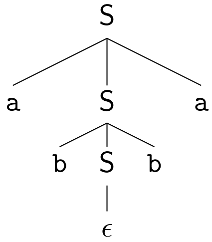
Yields

Definition (Yields)

The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree is called the **yield** of the parse tree.



Its yield is $N{*}X{+}N$.



Its yield is `abba`.

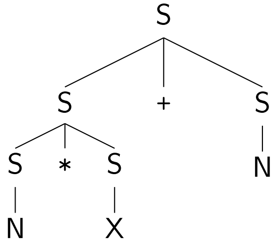
Relationship between Parse Trees and Derivations

Theorem (Parse Trees and Derivations)

For a given CFG $G = (V, \Sigma, S, P)$, for any sequence of variables or symbols $w \in (V \cup \Sigma)^*$:

$$S \Rightarrow^* w \iff \exists \text{ parse tree } T. \text{ s.t. } T \text{ yields } w$$

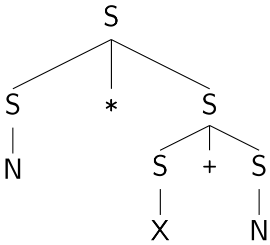
$$\begin{aligned} S &\Rightarrow S{+}S \\ &\Rightarrow S{*}S{+}S \\ &\Rightarrow N{*}S{+}S \\ &\Rightarrow N{*}X{+}S \\ &\Rightarrow N{*}X{+}N \end{aligned}$$

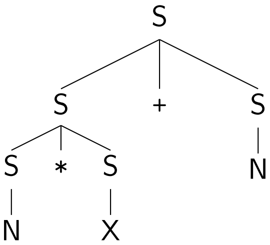


Ambiguous Grammars

$$
\begin{aligned}
S &\rightarrow N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

Actually, there are two distinct parse trees for a sentential form $N*X+N$:





Ambiguous Grammars

Definition (Ambiguous Grammar)

A **context-free grammar** $G = (V, \Sigma, S, P)$ is **ambiguous** if there exist a word $w \in \Sigma^*$ and two distinct parse trees whose yields are w. If not, G is **unambiguous**.

Theorem

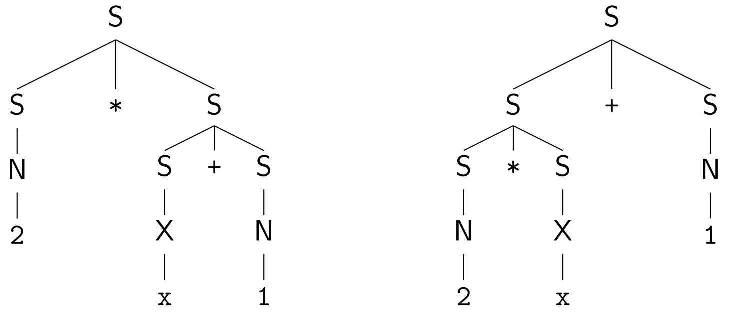
Let $G = (V, \Sigma, S, P)$ be a CFG. Then, the following numbers are equal for any sequence of variables or symbols $w \in (V \cup \Sigma)^*$:

1. *The number of parse trees whose yields are w.*
2. *The number of left-most derivations for w.*
3. *The number of right-most derivations for w.*

Ambiguous Grammars – Example

$$
\begin{aligned}
S &\rightarrow N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

This grammar is **ambiguous** because there are **two** parse trees for `2 * x + 1`:



$$
\begin{aligned}
S &\to N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\to \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\to \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

There are **two** left-most derivations for `2 * x + 1`:

1. Applying the production rule $S \to S\texttt{*}S$ first:

$$
\begin{aligned}
S &\overset{\text{lm}}{\Longrightarrow} S\texttt{*}S \overset{\text{lm}}{\Longrightarrow} N\texttt{*}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*}S\texttt{+}S \\
&\overset{\text{lm}}{\Longrightarrow} \texttt{2*}X\texttt{+}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+}N \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+1}
\end{aligned}
$$

2. Applying the production rule $S \to S\texttt{+}S$ first:

$$
\begin{aligned}
S &\overset{\text{lm}}{\Longrightarrow} S\texttt{+}S \overset{\text{lm}}{\Longrightarrow} S\texttt{*}S\texttt{+}S \overset{\text{lm}}{\Longrightarrow} N\texttt{*}S\texttt{+}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*}S\texttt{+}S \\
&\overset{\text{lm}}{\Longrightarrow} \texttt{2*}X\texttt{+}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+}S \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+}N \overset{\text{lm}}{\Longrightarrow} \texttt{2*x+1}
\end{aligned}
$$

Eliminating Ambiguity

Unfortunately,

- There is **NO** general algorithm to determine a CFG is ambiguous.
- There is **NO** general algorithm to remove ambiguity from a CFG.

Eliminating Ambiguity

Unfortunately,

- There is **NO** general algorithm to determine a CFG is ambiguous.
- There is **NO** general algorithm to remove ambiguity from a CFG.

Fortunately, there are well-known techniques to **eliminate** the ambiguity in a grammar commonly used in programming languages.

Eliminating Ambiguity

Unfortunately,

- There is **NO** general algorithm to determine a CFG is ambiguous.
- There is **NO** general algorithm to remove ambiguity from a CFG.

Fortunately, there are well-known techniques to **eliminate** the ambiguity in a grammar commonly used in programming languages.

$$
\begin{array}{rcl}
S & \rightarrow & N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

Eliminating Ambiguity

Unfortunately,

- There is **NO** general algorithm to determine a CFG is ambiguous.
- There is **NO** general algorithm to remove ambiguity from a CFG.

Fortunately, there are well-known techniques to **eliminate** the ambiguity in a grammar commonly used in programming languages.

$$
\begin{array}{lcl}
S & \rightarrow & N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

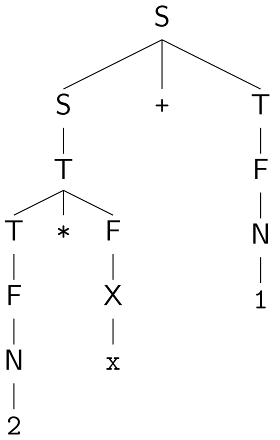
An equivalent but unambiguous grammar:

$$
\begin{array}{lcl}
S & \rightarrow & T \mid S\texttt{+}T \\
T & \rightarrow & F \mid T\texttt{*}F \\
F & \rightarrow & N \mid X \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

Eliminating Ambiguity

Now, the unique parse tree for `2 * x + 1` is:

$$
\begin{aligned}
S &\to T \mid S\texttt{+}T \\
T &\to F \mid T\texttt{*}F \\
F &\to N \mid X \mid \texttt{(}S\texttt{)} \\
N &\to \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\to \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$



Eliminating Ambiguity

First, analyze why the original grammar is ambiguous.

$$
\begin{aligned}
S &\rightarrow N \mid X \mid S\texttt{+}S \mid S\texttt{*}S \mid \texttt{(}S\texttt{)} \\
N &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}
$$

- The **precedence** of + and * is not specified.
  - For example, two parse trees for `1 * 2 + 3` interpreted as:

    `1 * (2 + 3)` and `(1 * 2) + 3`

  - Let's give * higher precedence than + to interpret it as `(1 * 2) + 3`.
- The **associativity** of + (or *) is not specified.
  - For example, two parse trees for `1 + 2 + 3` interpreted as:

    `1 + (2 + 3)` and `(1 + 2) + 3`

  - Let's give the left-associativity to + to interpret it as `(1 + 2) + 3`.

Eliminating Ambiguity – Precedence

To enforce the **precedence**, define new variables F for factors and T for terms:

Eliminating Ambiguity – Precedence

To enforce the **precedence**, define new variables F for factors and T for terms:

- A **factor** is a number, a variable, or a parenthesized expression:

  $$\texttt{42}, \quad \texttt{x}, \quad \texttt{(1 + 2)}, \quad \cdots$$

  In the grammar, F is defined as:

  $$F \rightarrow N \mid X \mid \texttt{(}S\texttt{)}$$

Eliminating Ambiguity – Precedence

To enforce the **precedence**, define new variables F for factors and T for terms:

- A **factor** is a number, a variable, or a parenthesized expression:

  $$\texttt{42}, \quad \texttt{x}, \quad \texttt{(1 + 2)}, \quad \cdots$$

  In the grammar, F is defined as:

  $$F \to N \mid X \mid (S)$$

- A **term** is the multiplication of one or more factors:

  $$\texttt{42}, \quad \texttt{2 * x}, \quad \texttt{2 * (1 + 2)}, \quad \texttt{1 * (x * y) * z}, \quad \cdots$$

  In the grammar, T is defined as:

  $$T \to F \mid T * F$$

Eliminating Ambiguity – Precedence

To enforce the **precedence**, define new variables F for factors and T for terms:

- A **factor** is a number, a variable, or a parenthesized expression:

  `42,   x,   (1 + 2),   ...`

  In the grammar, F is defined as:

  $$F \to N \mid X \mid (S)$$

- A **term** is the multiplication of one or more factors:

  `42,   2 * x,   2 * (1 + 2),   1 * (x * y) * z,   ...`

  In the grammar, T is defined as:

  $$T \to F \mid T*F$$

- An **expression** is the addition of one or more terms:

  `42,   1 + 2,   1 + 2 * 3,   (1 + 2) * 3 + 4),   ...`

  In the grammar, S is defined as:

  $$S \to T \mid S+T$$

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

$$
\begin{array}{rcl}
S & \rightarrow & T \mid S\texttt{+}T \\
T & \rightarrow & F \mid T\texttt{*}F \\
F & \rightarrow & N \mid X \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

$$
\begin{array}{lcl}
S & \rightarrow & T \mid S+T \\
T & \rightarrow & F \mid T*F \\
F & \rightarrow & N \mid X \mid (S) \\
N & \rightarrow & 0 \mid \cdots \mid 9 \mid 0N \mid \cdots \mid 9N \\
X & \rightarrow & \mathtt{a} \mid \cdots \mid \mathtt{z}
\end{array}
$$

- This grammar supports the **left-associativity** of + and $*$. How?

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

$$
\begin{array}{rcl}
S & \rightarrow & T \mid S\texttt{+}T \\
T & \rightarrow & F \mid T\texttt{*}F \\
F & \rightarrow & N \mid X \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

- This grammar supports the **left-associativity** of + and $*$. How?
- How to support the **right-associativity** of + and $*$?

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

$$
\begin{array}{lcl}
S & \rightarrow & T \mid S\texttt{+}T \\
T & \rightarrow & F \mid T\texttt{*}F \\
F & \rightarrow & N \mid X \mid \texttt{(}S\texttt{)} \\
N & \rightarrow & \texttt{0} \mid \cdots \mid \texttt{9} \mid \texttt{0}N \mid \cdots \mid \texttt{9}N \\
X & \rightarrow & \texttt{a} \mid \cdots \mid \texttt{z}
\end{array}
$$

- This grammar supports the **left-associativity** of + and $*$. How?
- How to support the **right-associativity** of + and $*$?

$$
\begin{array}{lcl}
S & \rightarrow & T \mid T\texttt{+}S \\
T & \rightarrow & F \mid F\texttt{*}T \\
\cdots
\end{array}
$$

Inherent Ambiguity

So far, we have discussed the **ambiguity** for grammars.
We will now discuss the **inherent ambiguity** for languages.

Definition (Inherent Ambiguity)

A language L is **inherently ambiguous** if all CFGs whose languages are L are ambiguous. (i.e. there is no unambiguous grammar for L)

For example, the following language is **inherently ambiguous**:

$$L = \{\mathtt{a}^i \mathtt{b}^j \mathtt{c}^k \mid i, j, k \geq 0 \land (i = j \lor j = k)\}$$

An example of ambiguous grammar for L is:

$$\begin{array}{lcl} S & \rightarrow & L \mid R \\ L & \rightarrow & A \mid L\mathtt{c} \\ A & \rightarrow & \epsilon \mid \mathtt{a}A\mathtt{b} \\ R & \rightarrow & B \mid \mathtt{a}R \\ B & \rightarrow & \epsilon \mid \mathtt{b}B\mathtt{c} \end{array}$$

Midterm Exam

- Midterm exam will be given in class.
- **Date:** 14:00-15:15 (1 hour 15 minutes), April 24 (Mon.).
- **Location:** 302, Aegineung (애기능생활관)
- **Coverage:** Lectures 1 – 13
- **Format:** short- or long-answer questions, including proofs
  - Closed book, closed notes
  - No questions about Scala code in the midterm exam.
- **Note that there is a lecture on April 26 (Wed.).**

Summary

1. Parse Trees
   - Definition
   - Yields
   - Relationship between Parse Trees and Derivations

2. Ambiguity
   - Ambiguous Grammars
   - Eliminating Ambiguity
   - Inherent Ambiguity

Next Lecture

- Pushdown Automata (PDA)

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr