Lecture 13 – Parse Trees and Ambiguity

COSE215: Theory of Computation

Jihyeok Park

PLRG

2025 Spring

Recall

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

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

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

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

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

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

- For a given word $w \in L(G)$, a **derivation** for w is $S \Rightarrow^* w$
- A sequence $\alpha \in (V \cup \Sigma)^*$ is a **sentential form** if $S \Rightarrow^* \alpha$.

Contents

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

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

Contents

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

2. Ambiguity
    Ambiguous Grammars
    Eliminating Ambiguity
    Inherent Ambiguity

Parse Trees

Consider the following CFG for balanced parentheses:

$$S \to \epsilon \mid (S) \mid SS$$

There are two different derivations for the sentential form $(S)(S)$:

Parse Trees

Consider the following CFG for balanced parentheses:

$$S \to \epsilon \mid (S) \mid SS$$

There are two different derivations for the sentential form $(S)(S)$:

$$① \quad S \;\Rightarrow_L\; SS \;\Rightarrow_L\; (S)S \;\Rightarrow\; (S)(S)$$

Parse Trees

Consider the following CFG for balanced parentheses:

$$S \to \epsilon \mid (S) \mid SS$$

There are two different derivations for the sentential form $(S)(S)$:

$$① \quad S \Rightarrow_L SS \Rightarrow_L (S)S \Rightarrow (S)(S)$$

$$② \quad S \Rightarrow_R SS \Rightarrow_R S(S) \Rightarrow (S)(S)$$

Parse Trees

Consider the following CFG for balanced parentheses:

$$S \to \epsilon \mid (S) \mid SS$$

There are two different derivations for the sentential form $(S)(S)$:

$$① \quad S \Rightarrow_L SS \Rightarrow_L (S)S \Rightarrow (S)(S)$$

$$② \quad S \Rightarrow_R SS \Rightarrow_R S(S) \Rightarrow (S)(S)$$

However, **parse trees** focus on the structure of the derivations instead of considering the order of the derivation steps.

Parse Trees

Consider the following CFG for balanced parentheses:

$$S \rightarrow \epsilon \mid (S) \mid SS$$

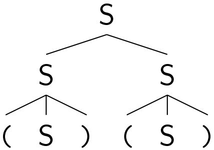
There are two different derivations for the sentential form $(S)(S)$:

$$① \quad S \Rightarrow_L SS \Rightarrow_L (S)S \Rightarrow (S)(S)$$

$$② \quad S \Rightarrow_R SS \Rightarrow_R S(S) \Rightarrow (S)(S)$$

However, **parse trees** focus on the structure of the derivations instead of considering the order of the derivation steps.

For example, the above two derivations have the same parse tree:



Parse Trees

Definition (Parse Trees)

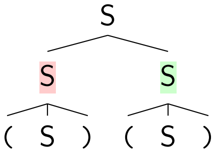
For a given CFG $G = (V, \Sigma, S, R)$, **parse trees** are trees satisfying:

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 $A \to X_1 X_2 \cdots X_k \in R$.
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: Balanced Parentheses

$$S \rightarrow \epsilon \mid (S) \mid SS$$

A parse tree for $(S)(S)$:



① $S \Rightarrow_L S\,S \Rightarrow_L (S)S \Rightarrow (S)(S)$

② $S \Rightarrow_R S\,S \Rightarrow_R S(S) \Rightarrow (S)(S)$

Parse Trees – Example 2: Even Palindromes

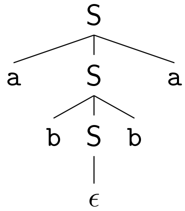
$$S \to \epsilon \mid \mathtt{a}S\mathtt{a} \mid \mathtt{b}S\mathtt{b}$$

A parse tree for `abba`:

Parse Trees – Example 2: Even Palindromes

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

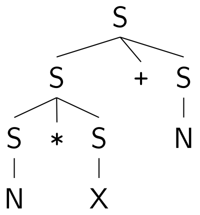
A parse tree for `abba`:



Parse Trees – Example 3: Arithmetic Expressions

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

A parse tree for $N\texttt{*}X\texttt{+}N$:



Yields

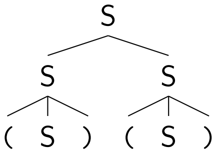
Definition (Yields)

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

Yields

Definition (Yields)

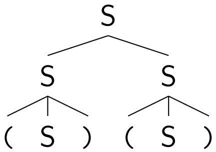
The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree from left to right is called the **yield** of the parse tree.



Yields

Definition (Yields)

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

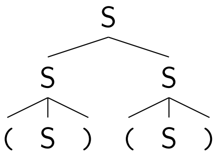


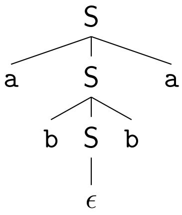
Its yield is $(S)\,(S)$.

Yields

Definition (Yields)

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



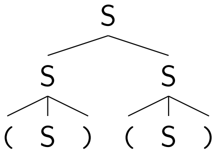


Its yield is (*S*)(*S*).

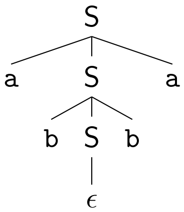
Yields

Definition (Yields)

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



Its yield is $(S)(S)$.

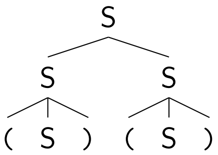


Its yield is abba.

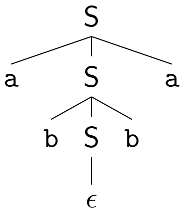
Yields

Definition (Yields)

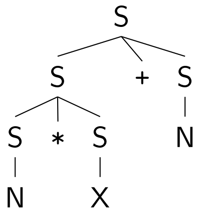
The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree from left to right is called the **yield** of the parse tree.



Its yield is $(S)(S)$.



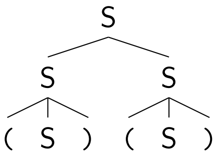
Its yield is abba.



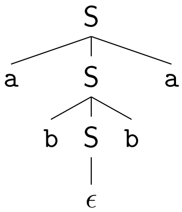
Yields

Definition (Yields)

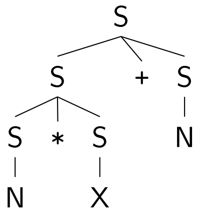
The sequence obtained by concatenating the labels (without ϵ) of the leaf nodes of a parse tree from left to right is called the **yield** of the parse tree.



Its yield is $(S)(S)$.



Its yield is abba.



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

Relationship between Parse Trees and Derivations

Theorem (Parse Trees and Derivations)

For a given CFG $G = (V, \Sigma, S, R)$, *for any sequence* $\alpha \in (V \cup \Sigma)^*$:

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

Relationship between Parse Trees and Derivations

Theorem (Parse Trees and Derivations)

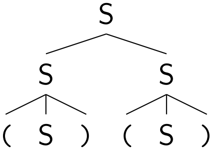
For a given CFG $G = (V, \Sigma, S, R)$, *for any sequence* $\alpha \in (V \cup \Sigma)^*$:

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

For example, consider the sequence $(S)(S)$:

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S)$$

$$\Updownarrow$$



Contents

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

2. Ambiguity
    Ambiguous Grammars
    Eliminating Ambiguity
    Inherent Ambiguity

Ambiguous Grammars

Is there always a **unique** parse tree for a given sentential form?

Ambiguous Grammars

Is there always a **unique** parse tree for a given sentential form?

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

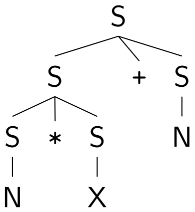
For example, consider the sentential form $N\texttt{*}X\texttt{+}N$:

Ambiguous Grammars

Is there always a **unique** parse tree for a given sentential form?

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

For example, consider the sentential form $N\texttt{*}X\texttt{+}N$:

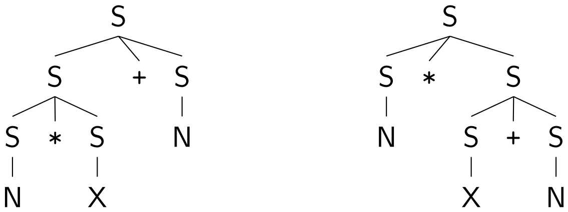


Ambiguous Grammars

Is there always a **unique** parse tree for a given sentential form?

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

For example, consider the sentential form $N\texttt{*}X\texttt{+}N$:



Actually, there are **two** parse trees for $N\texttt{*}X\texttt{+}N$.

Ambiguous Grammars

Definition (Ambiguous Grammar)

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

Ambiguous Grammars

Definition (Ambiguous Grammar)

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

Theorem

Let $G = (V, \Sigma, S, R)$ 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

PLRG

Definition (Ambiguous Grammar)

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

Theorem

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

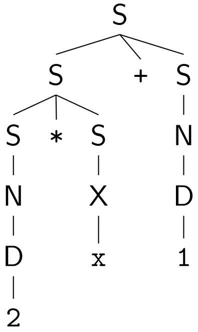
1. *The number of parse trees whose yields are w.*
2. *The number of left-most derivations for w.*
3. *The number of right-most derivations for w.*

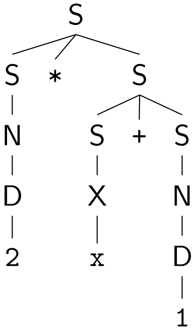
Proof) We can convert a left-most (or right-most) derivation for a word w into the corresponding parse tree for w and vice versa.

Ambiguous Grammars – Example

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

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

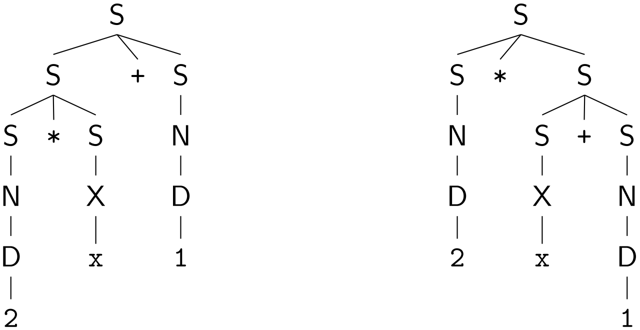




Ambiguous Grammars – Example

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

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



So, there are **two** left-most (or right-most) derivations for `2 * x + 1`.

Ambiguous Grammars – Example

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

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

1. Applying the production rule $S \rightarrow S\texttt{+}S$ first:

$$\begin{aligned} S &\Rightarrow_L S\texttt{+}S \Rightarrow_L S\texttt{*}S\texttt{+}S \Rightarrow_L N\texttt{*}S\texttt{+}S \Rightarrow_L D\texttt{*}S\texttt{+}S \Rightarrow_L \texttt{2*}S\texttt{+}S \\ &\Rightarrow_L \texttt{2*}X\texttt{+}S \Rightarrow_L \texttt{2*x+}S \Rightarrow_L \texttt{2*x+}N \Rightarrow_L \texttt{2*x+}D \Rightarrow_L \texttt{2*x+1} \end{aligned}$$

2. Applying the production rule $S \rightarrow S\texttt{*}S$ first:

$$\begin{aligned} S &\Rightarrow_L S\texttt{*}S \Rightarrow_L N\texttt{*}S \Rightarrow_L D\texttt{*}S \Rightarrow_L \texttt{2*}S \Rightarrow_L \texttt{2*}S\texttt{+}S \\ &\Rightarrow_L \texttt{2*}X\texttt{+}S \Rightarrow_L \texttt{2*x+}S \Rightarrow_L \texttt{2*x+}N \Rightarrow_L \texttt{2*x+}D \Rightarrow_L \texttt{2*x+1} \end{aligned}$$

Eliminating Ambiguity

Unfortunately,

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

Eliminating Ambiguity

Unfortunately,

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

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

Eliminating Ambiguity

Unfortunately,

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

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

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

Eliminating Ambiguity

Unfortunately,

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

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

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

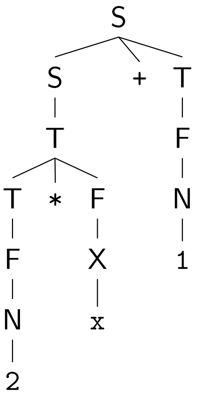
For example, an equivalent but unambiguous grammar is:

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

Eliminating Ambiguity

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

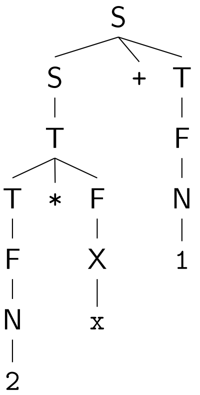
$$
\begin{aligned}
S &\rightarrow T \mid S\texttt{+}T \\
T &\rightarrow F \mid T\texttt{*}F \\
F &\rightarrow N \mid X \mid \texttt{(}S\texttt{)} \\
N &\rightarrow D \mid DN \\
D &\rightarrow \texttt{0} \mid \cdots \mid \texttt{9} \\
X &\rightarrow \texttt{a} \mid \cdots \mid \texttt{z}
\end{aligned}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



Eliminating Ambiguity

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

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



Let's try to understand how to eliminate the ambiguity in the original grammar.

Eliminating Ambiguity

First, analyze why the original grammar is ambiguous.

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

Eliminating Ambiguity

First, analyze why the original grammar is ambiguous.

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

- **Precedence** is not specified between different operators (+ and *).
  - 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`.

Eliminating Ambiguity

First, analyze why the original grammar is ambiguous.

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

- **Precedence** is not specified between different operators (`+` and `*`).
  - 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`.
- **Associativity** for the same operator (`+` or `*`).
  - 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:

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

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:

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

- 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:

  $$T \rightarrow F \mid T * F$$

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:

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

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

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

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

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

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

- This grammar supports the **left-associativity** of + and *. Why?
  - $S \rightarrow S\texttt{+}T$ and $T \rightarrow T\texttt{*}F$ are **left-recursive**.

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

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

- This grammar supports the **left-associativity** of + and *. Why?
    - $S \to S\texttt{+}T$ and $T \to T\texttt{*}F$ are **left-recursive**.
- Then, how to support the **right-associativity** of + and *?

Eliminating Ambiguity – Associativity

The unambiguous grammar is:

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

- This grammar supports the **left-associativity** of + and $*$. Why?
  - $S \to S\texttt{+}T$ and $T \to T\texttt{*}F$ are **left-recursive**.
- Then, how to support the **right-associativity** of + and $*$?
  - Replace the **left-recursive** rules with **right-recursive** rules!

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

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)

[1] https://en.wikipedia.org/wiki/Ogden's_lemma

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**:

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

[1] `https://en.wikipedia.org/wiki/Ogden's_lemma`

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**:

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

An example of ambiguous grammar for L is:

$$S \to L \mid R \quad L \to X \mid L\mathtt{c} \quad R \to Y \mid \mathtt{a}R$$
$$X \to \epsilon \mid \mathtt{a}X\mathtt{b} \quad Y \to \epsilon \mid \mathtt{b}Y\mathtt{c}$$

[1] https://en.wikipedia.org/wiki/Ogden's_lemma

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**:

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

An example of ambiguous grammar for L is:

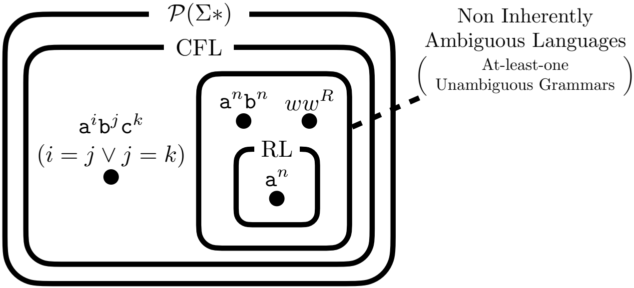
$$\begin{array}{llll} S \rightarrow L \mid R & L \rightarrow X \mid L\mathtt{c} & R \rightarrow Y \mid \mathtt{a}R \\ & X \rightarrow \epsilon \mid \mathtt{a}X\mathtt{b} & Y \rightarrow \epsilon \mid \mathtt{b}Y\mathtt{c} \end{array}$$

While we can prove that L is inherently ambiguous using the Ogden's lemma[1], we will not discuss it in this course.

[1] https://en.wikipedia.org/wiki/Ogden's_lemma

Summary

1. Parse Trees
   - Definition
   - Yields
   - Relationship between Parse Trees and Derivations
2. Ambiguity
   - Ambiguous Grammars
   - Eliminating Ambiguity
   - Inherent Ambiguity



Midterm Exam

- The midterm exam will be given in class.
- **Date:** 13:30-14:45 (1 hour 15 minutes), April 23 (Wed.).
- **Location:** 301, Aegineung (애기능생활관 301호)
- **Coverage:** Lectures 1 – 13
- **Format:** 7–9 questions with closed book and closed notes
  - Filling blanks in some tables, sentences, or expressions.
  - Construction of automata or grammars for given languages.
  - Proofs of given statements related to languages and automata.
  - Yes/No questions about concepts in the theory of computation.
  - etc.
- Note that there is **no class** on **April 28 (Mon.)**.
- Please refer to the **previous exams** in the course website:

https://plrg.korea.ac.kr/courses/cose215/

Next Lecture

- Pushdown Automata (PDA)

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