Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\mathbf{id} + \mathbf{id} * \mathbf{id}$$

Now consider the following grammar:

$E \rightarrow E + T | E - T | T$

$T \rightarrow T * F | T / F | F$

$F \rightarrow (E) | \mathbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\mathbf{id} + \mathbf{id} * \mathbf{id}$$

Now consider the following grammar:

$E \rightarrow E + T | E - T | T$
$T \rightarrow T * F | T / F | F$
$F \rightarrow (E) | \mathbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\textbf{id} + \textbf{id} * \textbf{id}$$

Now consider the following grammar:
$E \rightarrow E + T | E - T | T$
$T \rightarrow T * F | T / F | F$
$F \rightarrow (E) | \textbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

Conditional statements

Consider the following grammar:

$$\begin{array}{lcl} \mathit{stmt} & \rightarrow & \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt} \\ & \mid & \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\ \mathit{stmt} \\ & \mid & \textbf{other} \end{array}$$

decide if the following sentence belongs to the generated language:

$$\textbf{if}\ E_1\ \textbf{then}\ \textbf{if}\ E_2\ \textbf{then}\ S_1\ \textbf{else}\ S_2$$

Exercises

Consider the grammar:

$$S \rightarrow SS + |SS * |a$$

and the string $aa + a*$

- Give the leftmost derivation for the string
- Give the rightmost derivation for the string
- Give a parse tree for the string
- Is the grammar ambiguous or unambiguous?
- Describe the language generated by this grammar?

Define grammars for the following languages:

- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ is palindrom}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ contains the same occurrences of 0 and 1}\}$

Exercises

Consider the grammar:

$$S \rightarrow SS + |SS * |a$$

and the string $aa + a*$

- Give the leftmost derivation for the string
- Give the rightmost derivation for the string
- Give a parse tree for the string
- Is the grammar ambiguous or unambiguous?
- Describe the language generated by this grammar?

Define grammars for the following languages:

- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ is palindrom}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ contains the same occurrences of 0 and 1}\}$

CF grammars are capable to describe the syntax of most, but not all, the programming languages. For instance, the requirement that identifiers must be declared before their usage cannot be expressed in CF grammars.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars.
- A language that only admits ambiguous grammars is called an [illegible], e.g. $\{a^n b^m c^k \mid n = m \text{ or } m = k, n, m, k \geq 0\}$
- A Turing machine cannot decide whether a context-free language is inherently ambiguous or not.

CF grammars are capable to describe the syntax of most, but not all, the programming languages. For instance, the requirement that identifiers must be declared before their usage cannot be expressed in CF grammars.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language, e.g. $\{a^n b^m c^k \mid n = m \text{ or } m = k; n, m, k \geq 0\}$
- A Turing machine cannot decide whether a context-free language is inherently ambiguous or not

CF grammars are capable to describe the syntax of most, but not all, the programming languages. For instance, the requirement that identifiers must be declared before their usage cannot be expressed in CF grammars.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language, e.g. $\{a^n b^m c^k \mid n = m \text{ or } m = k; n, m, k \geq 0\}$
- A Turing machine cannot decide whether a context-free language is inherently ambiguous or not

CF grammars are capable to describe the syntax of most, but not all, the programming languages. For instance, the requirement that identifiers must be declared before their usage cannot be expressed in CF grammars.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language, e.g. $\{a^n\, b^m\, c^k \mid n = m \text{ or } m = k; n, m, k \geq 0\}$
- A Turing machine cannot decide whether a context-free language is inherently ambiguous or not

CF grammars are capable to describe the syntax of most, but not all, the programming languages. For instance, the requirement that identifiers must be declared before their usage cannot be expressed in CF grammars.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language, e.g. $\{a^n b^m c^k \mid n = m \text{ or } m = k; n, m, k \geq 0\}$
- A Turing machine cannot decide whether a context-free language is inherently ambiguous or not

ToC

ToC

Left Recursion

Left recursive grammars

A grammar $\mathscr{G}$ is left recursive if it has a non terminal A such that there is a derivation $A \stackrel{*}{\Longrightarrow} A\alpha$ for some sting α. Top-down parsing strategies cannot handle left-recursive grammars

Immediate left recursion

A grammar as an immediate left recursion if there is at least one production of the form $A \to A\alpha$. It is possible to transform the grammar still generating the same language and removing the left recursion. Consider the generale case:

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

where $n, m \geq 1$ and all β_i do not start with A. Equivalent productions are:

$$\begin{array}{lcl} A & \to & \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' & \to & \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

Left Recursion

Left recursive grammars

A grammar $\mathscr{G}$ is left recursive if it has a non terminal A such that there is a derivation $A \stackrel{*}{\Longrightarrow} A\alpha$ for some sting α. Top-down parsing strategies cannot handle left-recursive grammars

Immediate left recursion

A grammar as an immediate left recursion if there is at least one production of the form $A \rightarrow A\alpha$. It is possible to transform the grammar still generating the same language and removing the left recursion. Consider the generale case:

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

where $n, m \geq 1$ and all β_i do not start with A. Equivalent productions are:

$$\begin{array}{lcl} A & \rightarrow & \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' & \rightarrow & \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$