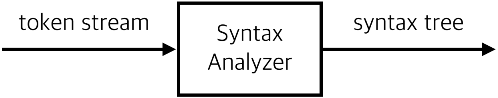
EC3204: Programming Languages and Compilers

Lecture 5 — Syntax Analysis (1): *Context-free Grammar*

Sunbeom So
Fall 2024

Roadmap for Building a Parser

A parser produces the grammatical structure of the source program.



1. **Specification**: how to express intended syntax?
   - E.g., `(1+3)` is syntatically valid, but `(1+3` is invalid.

   $\Rightarrow$ context-free grammar (CFG)

2. **Parsing**: given a sequence of tokens s, how to produce a syntax tree if $s \in L(CFG)$?

   $\Rightarrow$ Top-down and bottom-up parsing algorithms

Context-Free Grammar

Definition (Context-Free Grammar)

A context-free grammar G is defined as a quadruple:

$$G = (V, T, S, P)$$

- V: a finite set of **variables** (nonterminals)
- T: a finite set of **terminal symbols** (tokens)
- $S \in V$: the start variable
- P: a finite set of productions. A production has the form $x \to y$, where $x \in V$ and $y \in (V \cup T)^*$.

cf) Context-sensitive grammar[1] has productions of the form $\alpha x \beta \to \alpha y \beta$, where $\alpha, \beta \in (V \cup T)^*$ are the contexts of x.

[1] https://en.wikipedia.org/wiki/Context-sensitive_grammar

Example: CFG of Arithmetic Expressions

$$G = (\{E\}, \{+, *, -, (,), \mathbf{id}\}, E, P)$$

where P consists of:

$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

Example: CFG of Arithmetic Expressions

$$G = (\{E\}, \{+, *, -, (,), \mathbf{id}\}, E, P)$$

where P consists of:

$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

Q. Consider the token sequence $s = \mathbf{id} * (\mathbf{id} + \mathbf{id})$. Is $s \in L(G)$?

Example: CFG of Arithmetic Expressions

$$G = (\{E\}, \{+, *, -, (,), \mathrm{id}\}, E, P)$$

where P consists of:

$$E \to E + E \mid E * E \mid -E \mid (E) \mid \mathrm{id}$$

Q. Consider the token sequence $s = \mathrm{id} * (\mathrm{id} + \mathrm{id})$. Is $s \in L(G)$?
A. The language $L(G)$ includes s, because s is **derived** from the start variable E as follows:

$$\begin{array}{rcl} E & \Rightarrow & E * E \\ & \Rightarrow & \mathrm{id} * E \\ & \Rightarrow & \mathrm{id} * (E) \\ & \Rightarrow & \mathrm{id} * (E + E) \\ & \Rightarrow & \mathrm{id} * (\mathrm{id} + E) \\ & \Rightarrow & \mathrm{id} * (\mathrm{id} + \mathrm{id}) \end{array}$$

During the derivation, we apply the productions to **rewrite** the variables. Hence, productions can be viewed as **rewriting rules**.

Definition ($\Rightarrow$: Derivation Relation – "derive in one step")

Let $G = (V, T, S, P)$ be a context-free grammar. Let $\alpha A \beta$ be a string of terminals and variables, where $A \in V$ and $\alpha, \beta \in (V \cup T)^*$. Let $A \rightarrow \gamma$ be a production in G. Then, we say $\alpha A \beta$ derives $\alpha \gamma \beta$, and write

$$\alpha A \beta \Rightarrow \alpha \gamma \beta.$$

Definition ($\Rightarrow^*$: Closure of $\Rightarrow$ – "derive in zero or more steps")

$\Rightarrow^*$ is a relation that represents zero or more steps of derivations:

- Basis: For any string α of terminals and variables, $\alpha \Rightarrow^* \alpha$.
- Induction: If $\alpha \Rightarrow^* \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \Rightarrow^* \gamma$.

In our previous example, we have the relation $E \Rightarrow^* \mathbf{id} * (\mathbf{id} + \mathbf{id})$.

Language of Grammar

Definition (Sentential Form)

If $G = (V, T, S, P)$ is a context-free grammar and $S \Rightarrow^* \alpha$, then any string $\alpha \in (V \cup T)^*$ is a sentential form.

Definition (Sentence)

A sentential form α without non-terminals ($\alpha \in T^*$) is a sentence.

Definition (Language of Grammar)

The language of a grammar G is the set of all sentences:

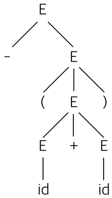
$$L(G) = \{w \mid S \Rightarrow^* w \wedge w \in T^*\}.$$

Parse Tree and Derivation

- A **parse tree** is a graphical tree-like representation of a derivation.
  - We can say: $L(G)$ is the set of all strings for which a corresponding parse tree can be constructed.
- For example, the derivation

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

is represented by the parse tree:



Each interior node and its children represent the application of a production.

  - The interior node is labeled by the head of the production.
  - The children are labeled by the symbols in the body of the production.

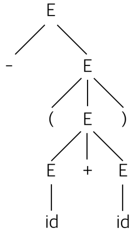
Parse Tree and Derivation

A parse tree ignores variations in the order in which symbols are replaced. Two derivations

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow \underline{-(\mathbf{id}+E)} \Rightarrow -(\mathbf{id}+\mathbf{id})$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow \underline{-(E+\mathbf{id})} \Rightarrow -(\mathbf{id}+\mathbf{id})$$

produce the same parse tree:



That is, there can be many-to-one relationships between derivations and parse trees; the parse trees are identical if the tow derivations use the same set of rules (they apply the same rules only in a different order).

Deriviations for Automatic Parsing

For automatic parsing, we consider two derivations.

- **Leftmost derivation**: the leftmost non-terminal in each sentential is always chosen. If $\alpha \Rightarrow \beta$ is a step in which the leftmost non-terminal in α is replaced, we write $\alpha \Rightarrow_l \beta$.

$$E \Rightarrow_l -E \Rightarrow_l -(E) \Rightarrow_l -(E+E) \Rightarrow_l -(\mathbf{id}+E) \Rightarrow_l -(\mathbf{id}+\mathbf{id})$$

- **Rightmost derivation** (canonical derivation): the rightmost non-terminal in each sentential is always chosen. If $\alpha \Rightarrow \beta$ is a step in which the rightmost non-terminal in α is replaced, we write $\alpha \Rightarrow_r \beta$.

$$E \Rightarrow_r -E \Rightarrow_r -(E) \Rightarrow_r -(E+E) \Rightarrow_r -(E+\mathbf{id}) \Rightarrow_r -(\mathbf{id}+\mathbf{id})$$

- If $S \Rightarrow_l^* \alpha$, α is a **left sentential form**.
- If $S \Rightarrow_r^* \alpha$, α is a **right sentential form**.

Deriviations for Automatic Parsing

For automatic parsing, we consider two derivations.

- **Leftmost derivation**: the leftmost non-terminal in each sentential is always chosen. If $\alpha \Rightarrow \beta$ is a step in which the leftmost non-terminal in α is replaced, we write $\alpha \Rightarrow_l \beta$.

$$E \Rightarrow_l -E \Rightarrow_l -(E) \Rightarrow_l -(E+E) \Rightarrow_l -(\mathbf{id}+E) \Rightarrow_l -(\mathbf{id}+\mathbf{id})$$

- **Rightmost derivation** (canonical derivation): the rightmost non-terminal in each sentential is always chosen. If $\alpha \Rightarrow \beta$ is a step in which the rightmost non-terminal in α is replaced, we write $\alpha \Rightarrow_r \beta$.

$$E \Rightarrow_r -E \Rightarrow_r -(E) \Rightarrow_r -(E+E) \Rightarrow_r -(E+\mathbf{id}) \Rightarrow_r -(\mathbf{id}+\mathbf{id})$$

- If $S \Rightarrow_l^* \alpha$, α is a **left sentential form**.
- If $S \Rightarrow_r^* \alpha$, α is a **right sentential form**.

Bad news: (even the same) derivations may yield different parse trees!

Ambiguity

A grammar is **ambiguous** if

- it produces more than one parse tree for some sentence,
- it produces multiple leftmost derivations for the same sentence, or
- it produces multiple rightmost derivations for the same sentence.

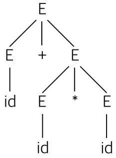
Example: Ambiguity

The grammar

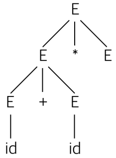
$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

is ambiguous, because it permits two different leftmost derivations for the sentence $\mathbf{id} + \mathbf{id} * \mathbf{id}$:

1. $E \Rightarrow E+E \Rightarrow \mathbf{id}+E \Rightarrow \mathbf{id}+E*E \Rightarrow \mathbf{id}+\mathbf{id}*E \Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}$



2. $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow \mathbf{id}+E*E \Rightarrow \mathbf{id}+\mathbf{id}*E \Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}$



Summary

- The syntax of a programming language is specified by context-free grammars.
- Basic definitions and terminologies: context-free grammar, derivation, left/rightmost derivations, parse tree, ambiguity

Next class: methods for eliminating the ambiguity, top-down parsing

cf) Regular Expression vs. Context-Free Grammar

- **Q**. Why not simply use regular expressions to specify the syntax?

cf) Regular Expression vs. Context-Free Grammar

- **Q**. Why not simply use regular expressions to specify the syntax?
- **A**. Programs often have recursive structures, which cannot be expressed using regex.
  - E.g., find a regular expression for describing the sum of integers enclosed by parentheses:

    $$(1+2), ((1+2)+3), (1+(2+3)), (((1+2)+3)+4), \cdots$$

    You can't! By contrast, context-free grammars can express the recursive structures:

    $$A \to n \in \mathbb{Z} \mid A_1 + A_2 \mid (A)$$

General fact: Every language that can be described by a regular expression can be described by a context-free grammar, but not vice versa.