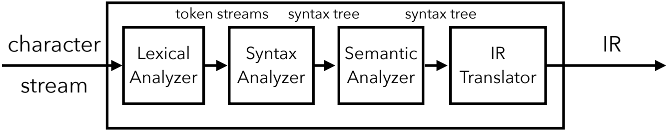
COSE312: Compilers

Lecture 3 — Syntax Analysis (1): Context-Free Grammar

Hakjoo Oh
2025 Spring

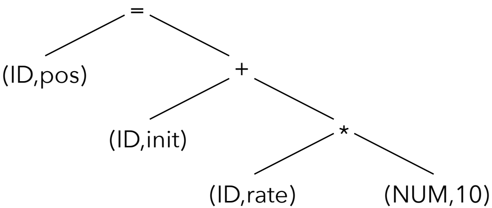
Syntax Analysis (Parsing)



Determine whether or not the input program is syntactically valid. If so, transform the stream

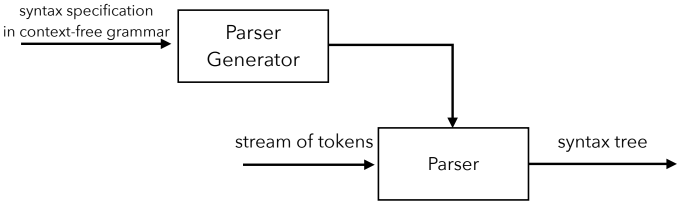
$$(\mathrm{ID}, \texttt{pos}), \texttt{=}, (\mathrm{ID}, \texttt{init}), \texttt{+}, (\mathrm{ID}, \texttt{rate}), *, (\mathrm{NUM}, \texttt{10})$$

into the syntax tree (or parse tree):



Contents

- **Specification**: context-free grammars.
- **Algorithms**: top-down and bottom-up parsing algorithms
- **Tools**: automatic parser generator



Context-Free Grammar

Example: Palindrome

- A string is a palindrome if it reads the same forward and backward.
- $L = \{w \in \{0,1\}^* \mid w = w^R\}$ ====
- L is not regular, but context-free.
- Every context-free language is defined by a recursive definition.
  - Basis: ϵ, 0, and 1 are palindromes.
  - Induction: If w is a palindrome, so are $0w0$ and $1w1$.
- The recursive definition is expressed by a context-free grammar.

$$
\begin{array}{lcl}
P & \rightarrow & \epsilon \\
P & \rightarrow & 0 \\
P & \rightarrow & 1 \\
P & \rightarrow & 0P0 \\
P & \rightarrow & 1P1
\end{array}
$$

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

  $$x \rightarrow y$$

  where $x \in V$ and $y \in (V \cup T)^*$.

Example: Expressions

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

where P:

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

The language includes $\mathbf{id} * (\mathbf{id} + \mathbf{id})$ because it is "derived" from E as follows:

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

Derivation

Definition (Derivation Relation, $\Rightarrow$)

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 $A \to \gamma$ is 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 $\Rightarrow$)

$\Rightarrow^*$ 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 $\alpha \Rightarrow^* \gamma$.

Language of Grammar

Definition (Sentential Forms)

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

Definition (Sentence)

A sentence of G is a sentential form with no non-terminals.

Definition (Language of Grammar)

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

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

Derivation is not unique

At each step in a derivation, there are multiple choices to be made, e.g., a sentence $-(\mathbf{id} + \mathbf{id})$ can be derived by

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

or alternatively by

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

Leftmost and Rightmost 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 $\alpha \Rightarrow_r \beta$.

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

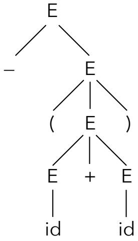
- If $S \Rightarrow_l^* \alpha$, α is a *left sentential form*.
- If $S \Rightarrow_r^* \alpha$, α is a *right sentential form*.

Parse Tree

A graphical tree-like representation of a derivation. E.g., the derivation

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

is represented by the parse tree:



- Each interior node represents the application of a production.
- The interior node is labeled by the head of the production.
- Children are labeled by the symbols in the body of the production.

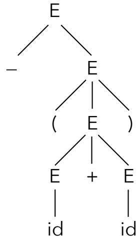
Parse Tree

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

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

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

produce the same parse tree:



The parse trees for two derivations are identical if the derivations use the same set of rules (they apply those rules only in a different order).

Ambiguity

A grammar is *ambiguous* if

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

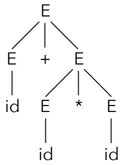
Example

The grammar

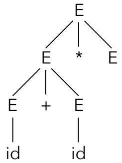
$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

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

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



Writing a Grammar

Transformations to make a grammar more suitable for parsing:

- eliminating ambiguity
- eliminating left-recursion
- left factoring

Eliminating Ambiguity

We can usually eliminate ambiguity by transforming the grammar. E.g., an ambiguous grammar:

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

To eliminate the ambiguity, we express in grammar

- (precedence) bind $*$ tighter than $+$
  - $1 + 2 * 3$ is always parsed by $1 + (2 * 3)$
- (associativity) $*$ and $+$ associate to the left
  - $1 + 2 + 3$ is always parsed by $(1 + 2) + 3$

An unambiguous grammar:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to \mathbf{id} \mid (E) \end{aligned}$$

- parse tree for $\mathbf{id} + \mathbf{id} + \mathbf{id}$
- parse tree for $\mathbf{id} + \mathbf{id} * \mathbf{id}$

Exercise

Transform the grammar

$$\begin{array}{l} E \to E + T \mid T \\ T \to T * F \mid F \\ F \to \mathbf{id} \mid (E) \end{array}$$

so that $*$ associate to the right.

Eliminating Left-Recursion

A grammar is left-recursive if it has a non-terminal A such that there A appears as the first right-hand-side symbol in an A-production, e.g.,

$$E \to E + T \mid T$$

To eliminate left-recursion, rewrite the grammar using right recursion:

$$\begin{array}{l} E \to T\ E' \\ E' \to +\ T\ E' \\ E' \to \epsilon \end{array}$$

In general, if $A \to A\alpha \mid \beta$ are two A-productions, we can eliminate left-recursion as follows:

$$\begin{array}{rcl} A & \to & \beta A' \\ A' & \to & \alpha A' \mid \epsilon \end{array}$$

Left Factoring

The grammar

$$\begin{array}{l} \boldsymbol{S \to \textbf{if } E \textbf{ then } S \textbf{ else } S} \\ \boldsymbol{S \to \textbf{if } E \textbf{ then } S} \end{array}$$

has rules with the same prefix. We can *left factor* the grammar as follows:

$$\begin{array}{l} \boldsymbol{S \to \textbf{if } E \textbf{ then } S\ X} \\ \boldsymbol{X \to \epsilon} \\ \boldsymbol{X \to \textbf{else } S} \end{array}$$

In general, if $\boldsymbol{A \to \alpha\beta_1 \mid \alpha\beta_2}$ are two $\boldsymbol{A}$-productions, we can refactor the grammar as follows:

$$\begin{array}{rcl} \boldsymbol{A} & \boldsymbol{\to} & \boldsymbol{\alpha A'} \\ \boldsymbol{A'} & \boldsymbol{\to} & \boldsymbol{\beta_1 \mid \beta_2} \end{array}$$

Exercise

Consider the grammar for regular expressions over a and b:

$$\begin{array}{rcl} rexpr & \rightarrow & rexpr + rterm \mid rterm \\ rterm & \rightarrow & rterm\ rfactor \mid rfactor \\ rfactor & \rightarrow & rfactor * \mid rprimary \\ rprimary & \rightarrow & a \mid b \end{array}$$

Find an equivalent, left-factored grammar without left-recursion.

Non-Context-Free Language Constructs

- Example 1: The problem of checking that identifiers are declared before they are used in a program:

$$L_1 = \{wcw \mid w \in (a|b)^*\}$$

  - E.g., $aabcaab$
  - The first w: the declaration of an identifier w
  - c: an intervening program fragment
  - The second w: the use of the identifier
- Example 2: The problem of checking that the number of formal parameters of a function agrees with the number of actual parameters in a call:

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1, m \geq 1\}$$

  - a^n and b^m represent the formal-parameter lists of two functions declared to have n and m arguments, respectively, while c^n and d^m are the actual-parameter lists in calls to these two functions.
- Checking these properties is usually done during the semantic-analysis

Summary

- The syntax of a programming language is usually specified by a context-free grammar.
  - derivation, left/rightmost derivations
  - parse trees
  - ambiguous/unambiguous grammars
  - grammar transformation (eliminating ambiguity, eliminating left-recursion, left factoring)