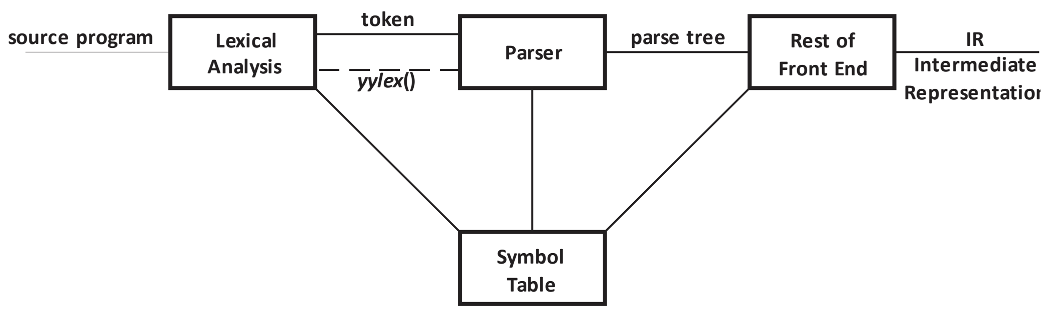


**Figure 1.** Position of the parser in the compiler model

**Context free grammars:**
A *context-free grammar* has four components:
1. A set of *tokens*, known as *terminal symbols* or *terminals*.
2. A set of *nonterminal symbols* or *noterminals*.
3. A set of productions where each production consists of a nonterminal symbol, called the *left side* of the production, an arrow, and a sequence of tokens and/or nonterminal symbols, called the *right side* of the production.
4. A designation of one of the *nonterminal* symbol as the *start* symbol.

Notation
1. Terminal symbols are expressed in **bold** print.
2. Nonterminal symbols are *italicized*.

**Example 1.** Write a grammar for an arbitrarily long expression consisting of single digits separated by either the plus sign or the minus sign.

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *list* | → | *list* **+** *digit* |
| 2 | *list* | → | *list* – *digit* |
| 3 | *list* | → | *digit* |
| 4 | *digit* | → | **0** |
| 5 | *digit* | → | **1** |
| 6 | *digit* | → | **2** |
| 7 | *digit* | → | **3** |
| 8 | *digit* | → | **4** |
| 9 | *digit* | → | **5** |
| 10 | *digit* | → | **6** |
| 11 | *digit* | → | **7** |
| 12 | *digit* | → | **8** |
| 13 | *digit* | → | **9** |

**Table 1.** Set of productions for the grammar of **Example 1.**

1. The set of terminal symbols (tokens), *T*={**+ - 0 1 2 3 4 5 6 7 8 9**}
2. The set of nonterminal symbols, *N*={*list digit*}

3. The set of productions *P*. Refer to table 1.
4. The starting nonterminal symbol *list*.

**Example 2.** Write a grammar for the language *micro*.

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *program* | → | **begin** *statement-list* **end** |
| 2 | *statement-list* | → | *statement* |
| 3 | *statement-list* | → | *statement-list* **;** *statement* |
| 4 | *statement* | → | **id :=** *expression* |
| 5 | *statement* | → | **read (** *id-list* **)** |
| 6 | *statement* | → | **write (** *expression-list* **)** |
| 7 | *id-list* | → | **id** |
| 8 | *id-list* | → | *id-list* **, id** |
| 9 | *expression-list* | → | *expression* |
| 10 | *expression-list* | → | *expression-list* **,** *expression* |
| 11 | *expression* | → | *primary* |
| 12 | *expression* | → | *expression additive-operator primary* |
| 13 | *primary* | → | **(** *expression* **)** |
| 14 | *primary* | → | **id** |
| 15 | *primary* | → | **intlit** |
| 16 | *additive-operator* | → | **+** |
| 17 | *additive-operator* | → | **-** |

**Table 2.** Set of productions for the *micro* grammar of **Example 2.**

1. The set of terminal symbols (tokens), *T*={**begin end read write id intlit ; := ( ) + -**}
2. The set of nonterminal symbols,
   *N*={*program statement-list statement id-list expression-list expression primary additive- operator*}
3. The set of productions *P*. Refer to table 2.
4. The starting nonterminal symbol *program*

**Example 3.** Write a grammar for expressions.

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *expression* | → | *expression* **+** *term* |
| 2 | *expression* | → | expression **–** term |
| 3 | *expression* | → | *term* |
| 4 | *term* | → | *term* ***** *factor* |
| 5 | *term* | → | *term* **/** *factor* |
| 6 | *term* | → | *factor* |
| 7 | *factor* | → | **(** *expression* **)** |
| 8 | *factor* | → | *id* |

**Table 3.** Set of productions expressions

1. The set of terminal symbols (tokens), *T*={ **id ( ) + - * /**}
2. The set of nonterminal symbols,
   *N*={*expression*, *term*, *factor* }
3. The set of productions *P*. Refer to table 3.
4. The starting nonterminal symbol *expression*.

**Example 4.** Write an abbreviated grammar for expressions.

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *E* | → | *E* **+** *T* |
| 2 | *E* | → | *E* **–** *T* |
| 3 | *E* | → | *T* |
| 4 | *T* | → | *T* ***** *F* |
| 5 | *T* | → | *T* **/** *F* |
| 6 | *T* | → | *F* |
| 7 | *F* | → | **(** *E* **)** |
| 8 | *F* | → | **id** |

**Table 3.** Set of productions expressions

1. The set of terminal symbols (tokens), *T*={ **id ( ) + - * /**}
2. The set of nonterminal symbols,
   *N*={*E*, *T*, *F* }
3. The set of productions *P*. Refer to table 3.
4. The starting nonterminal symbol *E*.

**Derivations**

Productions are rewriting rules. Beginning with the start symbol, each rewriting step replaces a nonterminal by the body of one of its productions.

Example: Consider the grammar of example 3 and derive **id+id*id**

| *Rule* | *left side* | | *Right side* |
|---|---|---|---|
| 1 | *E* | → | *E* **+** *T* |
| 4 | | → | *E* **+** *T* ***** *F* |
| 8 | | → | *E* + *T* * **id** |
| 6 | | → | *E* + *F* * **id** |
| 8 | | → | *E* + **id** * **id** |
| 3 | | → | *T* + **id** * **id** |
| 6 | | → | *F* + **id** * **id** |
| 8 | | → | **id** + **id** * **id** |

**Table 4.** Rightmost derivation of **id+id*id** from *E*

Consider $\alpha A\beta$ where $\alpha$ and $\beta$ are strings of grammar symbols that can include both terminal and nonterminal symbols. $A$ is a nonterminal symbol. Suppose $A \rightarrow \gamma$ is a production. We write $\alpha A\beta \Rightarrow \alpha\gamma\beta$. The symbol ⇒ means "derives in one step." When $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ rewrites $\alpha_1$ to $\alpha_n$ we say $\alpha_1$ derives $\alpha_n$. The symbol $\overset{*}{\Rightarrow}$ means "derives in zero or more steps." Likewise the symbol $\overset{+}{\Rightarrow}$ means "derives in one or more steps."

1. $\alpha \overset{*}{\Rightarrow} \alpha$, for any string $\alpha$.
2. If $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \overset{*}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$.

**Derivation order.**

1. $\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$ In leftmost derivations, the leftmost nonterminal in each sentential form is always chosen. Parsers that employ leftmost derivations are top-down and often use recursion. Such parsers are called **LL** meaning **L**eft-to-right scan of the input source and **L**eftmost derivations.
2. $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$ In rightmost derivations, the rightmost nonterminal in each sentential form is always chosen. Parsers that employ rightmost derivations are bottom-up or **LR** parsers for **L**eft-to-right scan of the input source and **R**ightmost derivation.

**Parser Trees and Derivations.**

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *E* | → | *E* **+** *E* |
| 2 | *E* | → | *E* ***** *E* |
| 3 | *E* | → | **-** *E* |
| 4 | *E* | → | **(** *E* **)** |
| 5 | *E* | → | **id** |

**Table 5.** Ambiguous grammar for expressions

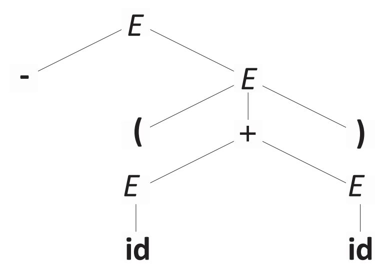


**Figure 2.** Parse tree for **–(id+id)**

| | *left side* | | *right side* |
|---|---|---|---|
| 3 | *E* | → | **-** *E* |
| 4 | | → | **- (** *E* **)** |
| 1 | | → | **- (***E* **+** *E***)** |
| 5 | | → | **- (***E* **+ id)** |
| 5 | | → | **- (id + id)** |

**Table 6.** Derivation for figure 2.

Ambiguity.

A grammar is ambiguous if there exists more than one parse tree for some sentence in the grammar. A grammar is ambiguous if there is more than one rightmost or leftmost derivation of a sentence in the grammar.

Consider the ambiguous grammar of Table 4 and the sentence **id+id*id**.

| | *left side* | | *right side* |
|---|---|---|---|
| 1 | *E* | → | *E* **+** *E* |
| 2 | *E* | → | *E* **+** *E* ***** *E* |
| 5 | *E* | → | *E* **+** *E* *** id** |
| 5 | *E* | → | *E* **+ id * id** |
| 5 | *E* | → | **id + id * id** |

**Table 7.** Rightmost derivation of **id+id*id** number 1

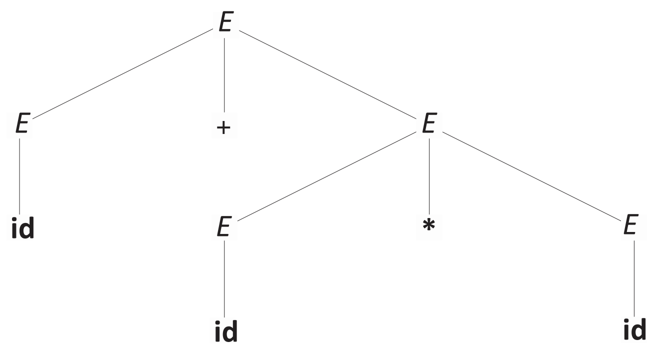


**Figure 3.** Rightmost derivation of **id+id*id** number 1.

| | *left side* | | *right side* |
|---|---|---|---|
| 2 | *E* | → | *E* ***** *E* |
| 5 | *E* | → | *E* *** id** |
| 1 | *E* | → | *E* **+** *E* *** id** |
| 5 | *E* | → | *E* **+ id * id** |
| 5 | *E* | → | **id + id * id** |

**Table 6.** Rightmost derivation of **id+id*id** number 2

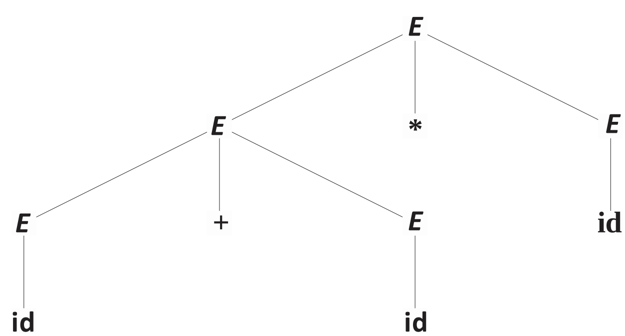


**Figure 4.** Rightmost derivation of **id+id*id** number 2.