Syntax Analysis: Context-free Grammars

Sequence of tokens is put together to form constructs in the language (expressions, declarations, statements...)

grammar: a formal specification of the syntax of a programming language.

Components of a Grammar:

- **terminal symbols** (tokens): the basic symbols from which strings of the language are formed.

- **nonterminal symbols**: syntactic variables that denote "related" sets of strings, e.g. statements, expressions.

- **start symbol**: a distinguished nonterminal; the set of strings it denotes is the language of the grammar.

- **productions**: rules specifying how strings can be derived in the grammar. It is of the form

\[ LHS \rightarrow RHS \]

where:

- \( LHS \) is a nonterminal; and
- \( RHS \) is a sequence of terminals and nonterminals.
**Example of a Context-Free Grammar:**

A grammar for arithmetic expressions:

- **terminals**: \{num, (, ), +, -, *, /\).

- **nonterminals**: \{E\}.

- **start symbol**: \(E\).

- **productions**:
  
  \[
  \begin{align*}
  E & \rightarrow E + E \\
  E & \rightarrow E - E \\
  E & \rightarrow E \times E \\
  E & \rightarrow E / E \\
  E & \rightarrow (E) \\
  E & \rightarrow \text{num}
  \end{align*}
  \]

**Derivations**

- Applying a production of a grammar can be thought of as a rewriting process, e.g.

  \[
  E + E \Rightarrow E + E \times E
  \]

  (“\(E + E\) rewrites to \(E + E \times E\)”) using the production ‘\(E \rightarrow E \times E\)’.

- A sequence of such rewritings is called a **derivation**.

**Example**:

\[
\begin{align*}
E & \Rightarrow E + E \\
& \Rightarrow E + E \times E \\
& \Rightarrow E + \text{num} \times E \\
& \Rightarrow E + \text{num} \times (E) \\
& \Rightarrow E + \text{num} \times (\text{num}) \\
& \Rightarrow \text{num} + \text{num} \times (\text{num})
\end{align*}
\]
**Derivation** – cont’d

**Leftmost Derivation**: a derivation where, at each step, only the leftmost nonterminal is replaced.

**Rightmost Derivation**: a derivation where, at each step, only the rightmost nonterminal is replaced.

**Example**:

**leftmost**: \[ E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow \text{num} + E * E \Rightarrow \text{num} + \text{num} * E \Rightarrow \text{num} + \text{num} * \text{num} \]

**rightmost**: \[ E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * \text{num} \Rightarrow E + \text{num} * \text{num} \Rightarrow \text{num} + \text{num} * \text{num} \]

**neither**: \[ E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow E + \text{num} * E \Rightarrow E + \text{num} * \text{num} \Rightarrow \text{num} + \text{num} + \text{num} \]

**Parse Trees**: Example:

Consider the derivation

\[ E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * (E) \]

The parse tree for this is:

- Internal nodes - nonterminals.
- Leaves - terminals.
- Parse tree for each string in the language should be unique.

⇒ Unique leftmost derivation
⇒ Unique rightmost derivation
**Ambiguity**

**Def:** A grammar is ambiguous if there is some sentence in its language for which there is more than one parse tree.

**Example:** The grammar

\[ E \rightarrow E + E \mid E \ast E \mid \text{id} \]

is ambiguous. E.g., the sentence

\[ \text{id} + \text{id} \ast \text{id} \]

has two parse trees:

![Parse Trees](image)

---

**Unambiguous Expression Grammar**

- **Precedence**
  - highest (**for**)  
  - next (**for** and **for**)  
  - lowest (**for** and **)  

- **Associativity**
  - right associativity (**for**)  
  - left associativity (**for** and **)  

\[
E \rightarrow E + T \mid E - T \mid T \\
T \rightarrow T * F \mid T / F \mid F \\
P \rightarrow P ** F \mid P \\
P \rightarrow -P \mid \text{ELEMENT} \\
\text{ELEMENT} \rightarrow (E) \mid \text{id} \mid \text{constant}
\]
Regular vs Context Free Grammars

Regular Grammar

- Rules are of the form:
  \[ A \rightarrow a \ B \]
  \[ C \rightarrow \varepsilon \]
- Cannot generate balanced parentheses.

Context Free Grammar for Balanced Parentheses

\[ S \rightarrow (S) \mid SS \mid \varepsilon \]

Types of Parsers

Top down Parsers

- Parse tree is constructed starting at the root.
- Leftmost derivation is found.

Bottom up Parsers

- Parse tree is constructed starting at the leaves.
- Rightmost derivation is found, in reverse.
4.3. FIRST and FOLLOW Sets

Definition [FIRST sets]:

- If $\alpha$ is any string of grammar symbols, then $\text{FIRST}(\alpha)$ is the set of terminals that begin strings derived from $\alpha$. I.e.,

$$\text{FIRST}(\alpha) = \{ a \mid a \text{ a terminal, } \alpha \Rightarrow^* a\beta \}$$

- If $\alpha \Rightarrow^* \varepsilon$ then $\varepsilon$ is also in $\text{FIRST}(\alpha)$. 
Example:

<table>
<thead>
<tr>
<th>Rule</th>
<th>FIRST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \rightarrow T \ E'$</td>
<td>$E$</td>
</tr>
<tr>
<td>$E' \rightarrow + \ T \ E' \mid \varepsilon$</td>
<td>$E'$</td>
</tr>
<tr>
<td>$T \rightarrow F \ T'$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T' \rightarrow * \ F \ T' \mid \varepsilon$</td>
<td>$T'$</td>
</tr>
<tr>
<td>$F \rightarrow ( \ E ) \mid \text{id}$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

Let $\$ be a special symbol (not a terminal or non-terminal) denoting end of input.

**Definition** [ FOLLOW sets ]:

- For any nonterminal $A$, FOLLOW($A$) is the set of terminals that can appear immediately to the right of $A$ in some sentential form.

- If $A$ can be the rightmost symbol in some sentential form, i.e., $S \Rightarrow^* \alpha A$, then $\$ is in FOLLOW($A$).
Example:

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E \rightarrow T \ E' )</td>
<td>{ (, id} }</td>
<td>( E \rightarrow { $, } )</td>
</tr>
<tr>
<td>( E' \rightarrow + T \ E' \mid \varepsilon )</td>
<td>{ +, \varepsilon }</td>
<td>( E' \rightarrow { $, } )</td>
</tr>
<tr>
<td>( T \rightarrow F \ T' )</td>
<td>{ (, id} }</td>
<td>( T \rightarrow { +, $, } )</td>
</tr>
<tr>
<td>( T' \rightarrow * F \ T' \mid \varepsilon )</td>
<td>{ *, \varepsilon }</td>
<td>( T' \rightarrow { +, $, } )</td>
</tr>
<tr>
<td>( F \rightarrow ( E ) \mid \text{id} )</td>
<td>{ (, id} }</td>
<td>( F \rightarrow { *, +, $, } )</td>
</tr>
</tbody>
</table>

**Computing FIRST Sets**

To compute \( \text{FIRST}(A) \) for any sequence of grammar symbols \( A \), apply the following rules until there is no change:

1. if \( A \) is a terminal then \( \text{FIRST}(A) = \{ A \} \).
2. if \( A \) is a nonterminal and has a production \( A \rightarrow \varepsilon \), then \( \varepsilon \) is in \( \text{FIRST}(A) \).
3. If \( A \) is a nonterminal and has a production
\[
A \rightarrow Y_1 \cdots Y_k,
\]
then:
- for \( i := 1 \) to \( k - 1 \) do
  - if \( \varepsilon \in \text{FIRST}(Y_i) \land \ldots \land \varepsilon \in \text{FIRST}(Y_k) \) and
    - a \((\neq \varepsilon)\) is in \( \text{FIRST}(Y_{i+1}) \)
  - then a is in \( \text{FIRST}(A) \).
- if \( \varepsilon \) is in each of \( \text{FIRST}(Y_1), \ldots, \text{FIRST}(Y_k) \)
  - then \( \varepsilon \) is in \( \text{FIRST}(A) \).

Example:

<table>
<thead>
<tr>
<th>FIRST(E)</th>
<th>FIRST(E')</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ (, id) }</td>
<td>{ (, id) }</td>
</tr>
<tr>
<td>{ +, \varepsilon }</td>
<td>{ +, \varepsilon }</td>
</tr>
<tr>
<td>{ (, id) }</td>
<td>{ (, id) }</td>
</tr>
<tr>
<td>{ *, \varepsilon }</td>
<td>{ *, \varepsilon }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIRST(F)</th>
<th>FIRST(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ (, id) }</td>
<td>{ (, id) }</td>
</tr>
<tr>
<td>{ (, id) }</td>
<td>{ (, id) }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIRST(F)</th>
<th>\text{FIRST}(F) = { (, id) }</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{FIRST}(T') = { *, \varepsilon }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(E') = { +, \varepsilon }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(E) = \text{FIRST}(F) = { (, id) }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(T) = \text{FIRST}(E) = { (, id) }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(E) = \text{FIRST}(T) = { (, id) }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(F) = \text{FIRST}(E) = { (, id) }</td>
<td></td>
</tr>
<tr>
<td>\text{FIRST}(T) = \text{FIRST}(E) = { (, id) }</td>
<td></td>
</tr>
</tbody>
</table>
Computing FOLLOW Sets

To compute FOLLOW sets, apply the following rules until there is no change:

1. $S$ is in FOLLOW($S$), where $S$ is the start symbol and $\$$ is the end-of-input marker.

2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST($\beta$) except $\epsilon$ is in FOLLOW($B$).

3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where $\epsilon \in$ FIRST($\beta$), then everything in FOLLOW($A$) is in FOLLOW($B$).
Example:

\[ E \rightarrow T \ E' \]
\[ E' \rightarrow T E' + \epsilon \]
\[ T \rightarrow F T' \]
\[ T' \rightarrow * F T' + \epsilon \]
\[ F \rightarrow ( E ) \mid \text{id} \]

**FOLLOW(E)**

- $\epsilon$ FOLLOW(E)
- F \rightarrow ( E ) \epsilon FOLLOW(E)
- FOLLOW(E) = \{ $, \) \}

**FOLLOW(E')**

- E \rightarrow T E' \& E' \rightarrow + T E'
- FOLLOW(E) is contained in FOLLOW(E')
- \{ $, ) \} is contained in FOLLOW(E')
- FOLLOW(E') = \{ $, ) \}

---

Example:

\[ E \rightarrow T E' \]
\[ E' \rightarrow + T E' + \epsilon \]
\[ T \rightarrow F T' \]
\[ T' \rightarrow * F T' + \epsilon \]
\[ F \rightarrow ( E ) \mid \text{id} \]

**FIRST**

- \{ (, id \}
- \{ +, \epsilon \}
- \{ (, id \}
- \{ *, \epsilon \}
- \{ (, id \}

**FOLLOW(T)**

- E \rightarrow T E' \& E' \rightarrow + T E'
- FIRST(E') – \{ \epsilon \} is contained in FOLLOW(T)
- \{ + \} is contained in FOLLOW(T)
- \epsilon belongs to FIRST(E')
- FOLLOW(E) is contained in FOLLOW(T)
- \{ $, ) \} is contained in FOLLOW(T)
- FOLLOW(T) = \{ +, $, ) \}

---
Example:

E → T E' 
E' → + T E' | ε 
T → F T' 
T' → * F T' | ε 
F → ( E ) | id 

<table>
<thead>
<tr>
<th>FIRST</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → { (, id}</td>
</tr>
<tr>
<td>E' → { +, ε}</td>
</tr>
<tr>
<td>T → { (, id}</td>
</tr>
<tr>
<td>T' → { *, ε}</td>
</tr>
<tr>
<td>F → { (, id}</td>
</tr>
</tbody>
</table>

FOLLOW(T') = { +, $, ) } 

T → F T' & T' → * F T' 
FIRST(T') - { ε } is contained in FOLLOW(F) 
\[ \rightarrow \{ * \} \text{ is contained in FOLLOW(F)} \n\] 
\[ \epsilon \text{ belongs to FIRST(T')} \n\] 
\[ \rightarrow \text{FOLLOW(T) is contained in FOLLOW(F)} \n\] 
\[ \rightarrow \{ +, $, ) \} \text{ is contained in FOLLOW(F)} \n\] 
FOLLOW(F) = { *, +, $, ) }
Example:

<table>
<thead>
<tr>
<th>Grammar</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → T E'</td>
<td>E { (, id}</td>
<td>E { $, )}</td>
</tr>
<tr>
<td>E' → + T E'</td>
<td>E' { +, ε}</td>
<td>E' { $, )}</td>
</tr>
<tr>
<td>T → F T'</td>
<td>T { (, id}</td>
<td>T { +, $, )}</td>
</tr>
<tr>
<td>T' → * F T'</td>
<td>T' { *, ε}</td>
<td>T' { +, $, )}</td>
</tr>
<tr>
<td>F → ( E )</td>
<td>F { (, id}</td>
<td>F { *, +, $, )}</td>
</tr>
</tbody>
</table>

CSE Department