Compilers Design

M. T. Bennani
Assistant Professor, FST - El Manar University, URAPOP-FST

-Academic Year 2020-2021-

Outline

- Recall the aim of the syntax analysis
- Context-Free Grammars
  - Formal Definition
  - Derivation
- Writing a Grammar
  - Eliminating Ambiguity
  - Elimination of Left Recursion
  - Left Factoring
- Generate Abstract Syntax Tree

Syntax analysis objective

- Regular languages: Widely used but are the weakest formal languages
  - Strings of balanced parentheses are not regular: $\{(^i)^i | i \geq 0\}$
  - Finite automaton can't remember # of times it has visited a particular state
- Another limit: Unable to handle function structure.

Goal: Distinguish between valid and invalid strings of tokens

- **Input:** Sequence of tokens from lexer
- **Output:** Parse tree of the program

Note:
Some parsers never produce a parse tree.

Formal Definition

Context-Free Grammar (Grammar)

It consists of terminals, nonterminals, a start symbol, and productions.

- *Terminals*: Synonyms of token names (Basic symbols from which strings are formed).
- *Nonterminals*: Syntactic variables that denote sets of strings. They are a composition of terminals and nonterminals. They impose a hierarchical structure on the language.
- *Start symbol*: A nonterminal symbol which denotes the language generated by the grammar.
- *Productions*: They specify the manner in which the terminals and nonterminals can be combined to create strings.

Formal Definition - Example

The following grammar (G_1):

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid id
\end{array}
$$

- Terminals: $+$,*,(,),id
- Nonterminals: E,T,F
- Start symbol: E
- Productions: E$\rightarrow$E$+$T, etc.

- Nonterminal is also called *head* of *left side* of the production
- Sometimes the symbol $::=$ has been used instead of $\rightarrow$
- A body of right side consists of zero or more terminals and nonterminals

Derivation

- The **construction of a parse tree** can be made by taking a **derivational view**, in which productions are treated as **rewriting rules**
- Beginning with the start symbol, each **rewriting step replaces a nonterminal by the body of one of its productions**
- **The derivational view** corresponds the **top-down construction of a parse tree**

Notes

- *leftmost* derivations: The leftmost nonterminal in each sentential is always chosen.
- *rightmost* derivations: The rightmost nonterminal in each sentential is always chosen.

Derivation - Example

- The following grammar (G_2):

$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid id$$

- The string (S_1): -(id + id) is a sentence of (G_2) because there is a derivation:

(D_1): E $\Rightarrow$ -E $\Rightarrow$ -(E) $\Rightarrow$ -(E + E) $\Rightarrow$ -(id + E) $\Rightarrow$ -(id + id)

- Generate the parse tree related to D_1

Notes

- D_1 is a leftmost derivation.
- (D_2): E $\Rightarrow$ -E $\Rightarrow$ -(E) $\Rightarrow$ -(E + E) $\Rightarrow$ -(E + id) $\Rightarrow$ -(id + id)
- D_2 is a rightmost derivation.

Draw the parse tree related to D_2 and compare it to the previous one.

Ambiguity

- A grammar that produces more than one parse tree for some sentence is said to be *ambiguous*.
- An amibiguous grammar is one that produces more than one leftmost derivation or more than one rightmost derivation for the same sentence.

Example

The sentence S_2: id $+$ id * id which belongs to G_2 shows that the grammar is ambiguous.

1. Compute a derivation of the sentence S_2 and generate its related parse tree
2. Compute a different derivation of the sentence S_2 and generate its related parse tree
3. Suggest a new definition of ambiguous grammar

Eliminating Ambiguity

- Stratification associates to each production at most one process
- If a grammar recognizes a sentence, only one derivation tree is generated

Example

The following grammar G_3: R $\rightarrow$ a | b | ... z | ε | R"|"R | R* | (R) is ambiguous because a | b* has two different meanings.

Steps

1. Build atomic sentences
2. Combine star's and atomic sentences
3. Concatenate star's sentences
4. Union of concatenated expressions

Eliminating Ambiguity - Example

1. Build atomic sentences
   A $\rightarrow$ a | b | ... | z
   A $\rightarrow$ ε
   A $\rightarrow$ (S)
2. Combine star's and atomic sentences S $\rightarrow$ A | S*
3. Concatenate star's sentences C $\rightarrow$ S | CS
4. Union of concatenated expressions E $\rightarrow$ C | E"|"C

Exercise

Eliminating Ambiguity of the grammar G_2

Elimination of Left Recursion

Left recursion

- A grammar is left recursive if it has a nonterminal A such that there is a derivation A $\Rightarrow$ Aa for some string a.
- Top-down parsing methods cannot handle left-recurcive grammars.

Basic Idea

$$A \to A\alpha \mid \beta \left\{ \begin{array}{lcr} A & \to & \beta\ B \\ B & \to & \alpha\ B \mid \varepsilon \end{array} \right.$$

Exercise

Translate G_1 to eliminate the recursion.

Left Factoring

Definition

- Left factoring a grammar is a transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing.
- When a choice between two alternative productions is not clear, we may be able to rewrite the productions to defer the decision until enough of the input has been seen that we can make the right choice.

Basic Idea

$$\mathrm{A} \rightarrow \alpha\beta_1|\alpha\beta_2|\gamma \left\{ \begin{array}{lll} \mathrm{A} & \rightarrow & \alpha\ \mathrm{B} \mid \gamma \\ \mathrm{B} & \rightarrow & \beta_1 \mid \beta_2 \end{array} \right.$$

Exercise

Left factoring the following grammar G_4:

E $\rightarrow$ **if** exp **then** stmt **else** stmt | **if** exp **then** stmt

Definitions

- Abstract Syntax Tree or (syntax tree), represents the hierarchical syntactic structure of the source program.
- Syntax tree (Abstract structure) is different from parse tree (Operational structure)

Example

$$
\begin{array}{lcl}
E & \rightarrow & T + E \mid T \\
E & \rightarrow & T \\
T & \rightarrow & \text{int} \\
T & \rightarrow & \text{int} * T \\
T & \rightarrow & (E)
\end{array}
$$

- $E1.val = T.val + E2.val$
- $E.val = T.val$
- $T.val = int.val$
- $T.val = int.val * T.val$
- $T.val = E.val$