LANGUAGE PROCESSORS

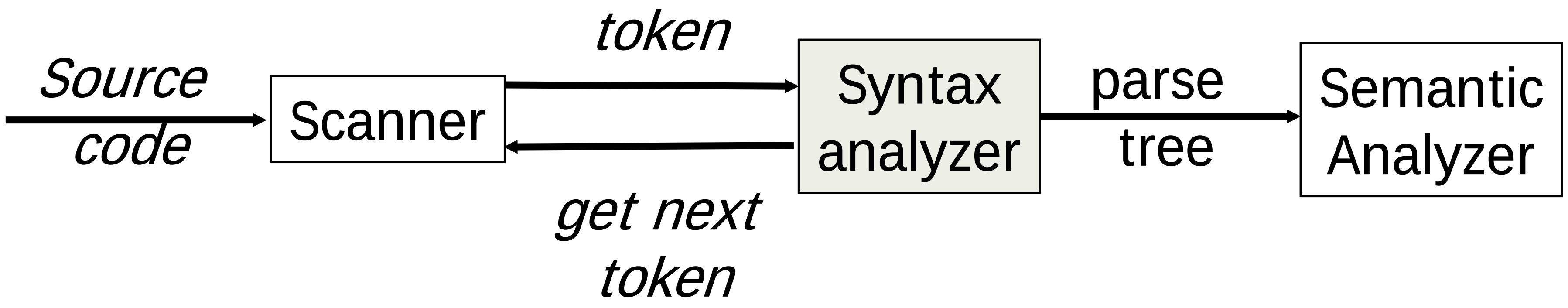
UNIT 4: SYNTAX ANALYSIS

David Griol Barres
dgriol@inf.uc3m.es
Computer Science Department
Carlos III University of Madrid
Leganés (Spain)

OUTLINE

- Position of the parser
- The role of the parser
- Advantages of using a grammar
- Types of syntax analyzers
  - Top-down parsing
  - Bottom-up parsing
- Syntax analyzers, Problems

Position of the parser



- **OBJECTIVE:** Determine the syntax structure of a program.
- **INPUT**: Sequence of tokens.
- **OUTPUT**: Parse tree.

The role of the parser

- The parser carries out:
  - obtains a string of tokens from the scanner,
  - verifies that the string can be generated by the grammar for the source language,
  - report syntax errors,
  - recover from common errors so that it can continue processing the remaining tokens.

Advantages of using a grammar

- It gives a precise syntactic specification of a programming language;
- efficient parsers can be constructed from certain classes of grammars;
- the parser construction process can reveal syntactic ambiguities;
- imparts a structure to a programming language that makes code generation and error detecting easier;
- it is easier to extend and modify the language.

Types of syntax analyzers

- **top-down parsers**
  - build parse trees from the top (root) to the bottom (leaves).
  - Predictive parsing.
    - recursive-descent parsing .
    - table-driven LL(1) parsing.
- **bottom-up parsers**
  - build parse trees from the leaves to the top.

Syntax analyzer

- For both top-down and bottom-up analysis
  - the input to the parser is scanned from left to right, one symbol at a time
  - they work on subclasses of grammars
- In general the grammars will be LL and LR
  - LR(k) $\supset$ LL(k)
  - In practice, only LR(1) and LL(1) are used
- Many compilers are *parser-driven*
- There are tools that automatically generate syntax analyzers (YACC, Bison)

Top-down parsing

- Begin with the start symbol and apply the productions until obtaining the desired string.

  1. the root is labeled by the start symbol
  2. while there are nonterminal symbols, carry out a left derivation

Top-down parsing

- Example
  - Input: Id.*.Id.+.Id
  - Grammar:

    Expression::=**Expression**.*.**Term** | **Expression** .+.**Term** | **Term**

    Term ::= Id | Number
  - Derivation:

    Expression → **Expression** .+.**Term** →

    Expression.*.**Term**.+.**Term** →

    Term.*.**Term**.+.**Term** →

    Id.*.**Term**.+.**Term** →

    Id.*.Id.+.**Term** → Id.*.Id.+.Id

Bottom-up parsing

- **handle**: longest sequence of symbols in the left-hand side of the input that can be found in the right-hand side of a production, and such that all the symbols to the right are terminals
  - Example:
    - input: **Expression**.*.**Term**.+.Id
    - handle: **Expression**.*.**Term**
- Algorithm
  1. start by considering an input stream
  2. try to get the axiom finding the handle and reducing it with the corresponding production

Bottom-up parsing

Grammar:

Expression::=Expression.*.Term | Expression .+.Term | Term

Term ::= Id | Number

Id.*.Id.+.Id → **Term**.*.Id.+.Id → **Expresion**.*.Id.+.Id →
Expresion.*.**Term**.+.Id → **Expresion**.+.Id →
Expresion.+.**Term** → **Expresion**

Syntax analyzers, Problems

- **Top-down:**
  - More than one option: A::= α | β
    - Backtracking.
    - analyze the next input elements.
  - Left recursion
    - Elimination of left recursion.
  - Ambiguity
    - left factoring.
- **Bottom-up:**
  - More than one option: A::= α and α is the handle.