Week3

Thursday, February 3, 2022 4:37 PM

Programming Languages

They can be classified by generation:
- 1st gen: machine languages
- 2nd gen: Assembly languages
- 3rd gen: Higher level languages, like Fortran, Cobol, Lisp, C, Java
- 4th gen: languages designed for specific tasks: NOMAD, SQL

Scripting Languages: interpreted languages w/ high level operators for gluing computations (scripts): JS, Perl, Python

1. Analysis Phase:
   - breaks up the source program and builds an intermediate representation.
   - Includes the syntax of the language
     - ↳ describes the proper form of its programs.

   semantics defines what the program means:
   - ↳ what each program does when executing,
   - the meaning of words.

2. The Synthesis Phase: translates intermediate code to the target program (executable).

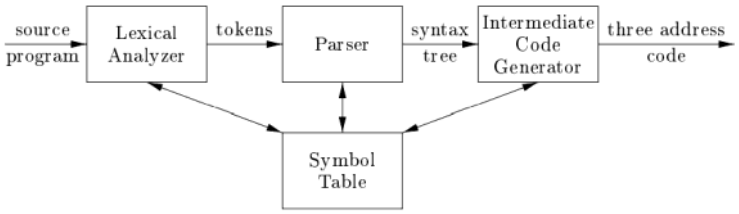

Figure 2.3 A model of a compiler front end

2.2 Syntax Definition

Free grammar / grammar:
- specifies the syntax of a language.
- Describes the hierachical structure of most programming language constructs

* Example: if else statement (proposición),

Statement → if (expression) {statement} else {statement}

- Statement → can have the form
- if, else → keywords

which is a structuring rule: stmt → if (expr) stmt else stmt
- ↳ such a rule is called a production

PRODUCTION: ✓ lexical elements (keyword like if or ()) are called TERMINALS

expr, stmt } are sequences of TERMINALS, but called NON TERMINALS.

2.2.1 Definition of Grammars

A context-free grammar has 4 components:

1. A set of terminal symbols, sometimes called tokens.

↳ the terminals are elementary symbols of the language defined by the GRAMMAR.

2. A set of nonterminals, sometimes called "syntactic variables"
   ↳ the nonterminal is a set of strings of terminals

3. A set of productions
   ↳ a production consists of a nonterminal, called the HEAD or LEFT SIDE of the production (BEGINNING), an arrow, and a sequence of terminals/nonterminals, called the BODY or RIGHT SIDE of the production.

   ↳ the intent of a production: specify one of the written forms of a CONSTRUCT.

   HEAD → CONSTRUCT
   BODY → a written form of the CONSTRUCT

4. The designation of one of the nonterminals as the START SYMBOL.

We specify GRAMMARS by listing their productions
↳ the productions for the start symbol go first.

TERMINALS: digits
< > =
while / for / if
} ANY NON-ITALIC SYMBOL

NON-TERMINALS: ITALIC NAMES

* Note: PRODUCTIONS with the same nonterminal as HEAD can have their bodies grouped, with alternative bodies separated by the symbol | (or).

* EXAMPLE: An expression consisting of digits and plus & minus signs (2.1)

a) 9 − 5 + 2
b) 3 − 1
c) 7

→ Since +/- signs appear between 2 digits, we call those expressions as LISTS OF DIGITS SEPARATED WITH +/- SIGNS.

→ The following grammar describes the syntax of these expr; The productions are:

list	→ *list* + *digit*	(2.1)
list	→ *list* − *digit*	(2.2)
list	→ *digit*	(2.3)
digit	→ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	(2.4)

(the bodies of the 3 productions with nonterminal list as head can be grouped)

list → *list* + *digit* | *list* − *digit* | *digit*

→ the TERMINALS of the GRAMMAR are the symbols:
+ − 0 1 2 3 4 5 6 7 8 9

→ the NONTERMINAL (italic) names like *list* / *digit*

→ We say a production is FOR a nonterminal if the nonterminal is head of such production.

↳ a string of terminals is a sequence of ZERO OR MORE terminals

2.2.2 Derivations

→ a GRAMMAR derives strings by:

1. beginning with the START SYMBOL,
2. and repeatedly replacing a NONTERMINAL by the BODY of a production for that NONTERMINAL.

→ The terminal strings that can be derived from the START SYMBOL form the LANGUAGE defined by the grammar.

* EXAMPLE (2.2): the language defined by the grammar of Example 2.1 consists of lists of digits separated by plus and minus signs.

↳ the 10 productions for the NONT digit allow it to have any TERMINAL of 0,1,2,...,9.

↳ From production (2.3) a single digit is a list

↳ Productions (2.1) and (2.2) express the rule that: "any list followed by a plus/minus and then a digit make up a list!"

↳ Productions (2.1) and (2.4) are ALL we need to define the desired language. I.e., we can deduce 9−5+2 is a *list* by:

a) 9 is a *list* by production (2.3), since 9 is a *digit*.

b) 9−5 is a *list* by production (2.2), since 9 is a *list* and 5 is a *digit*.

c) 9−5+2 is a *list* by production (2.1), since 9−5 is a *list* and 2 is a *digit*.

TOKENS VS TERMINALS

→ lexical analyzer reads the characters from the source, groups them into lexemes, and outputs tokens that represent these lexemes.

→ TOKEN: 2 components: <token name, attribute value>

→ token names are called TERMINALS. So TOKEN = TERMINALS.

→ the attr value is a pointer to the symbol Table place for the token's info. NOT PART OF GRAMMAR