프로그래밍언어의 개념

Concepts of Programming Language

(Lecture 05 : Chapter 3 - Describing Syntax and Semantics)

Prof. A. P. Shrestha, Ph.D.

Dept. of Computer Science and Engineering, Sejong University

Last Class

Chapter 2 - Evolution of Major Programming Languages

- ALGOL
- Logic Based Programming Language-Prolog
- Object Oriented Programming Language
- Scripting Languages

Today

Chapter 3 - Describing Syntax and Semantics

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax

Next class

Chapter 3- Describing Syntax and Semantics

5.1.1 Introduction

- The study of programming languages is somewhat similar to the study of natural languages

Natural Language

Korean Language
최씨는 밥을 먹었다.

English Language
Mr. Choi ate Rice

Programming Language

Python

```
for name in my_list:
        if name in invited_people:
                print name
```

Scheme

```
(map (lambda (name) (if (cond ((member name invited_people) name)) (display name) name)) my_list)
```

Q) What is difference between these two languages (both in natural and programming languages)?

Ans→ Each language conveys same meaning but uses different grammatical structure

5.1.2 Description of a Language

Q) Are there any mistakes in the following three sentences?

1. Fed I the dog.
2. I fed the wall.
3. I fed the dog.

Grammar rules and meaning!!

5.1.3 Syntax and Semantics

- The study of programming languages, like the study of natural languages, can be divided into examinations of

  - syntax, and

  - semantics.

- ***Syntax:*** the form of its expressions, statements, and program units.

- ***Semantics*** is the meaning of those expressions, statements, and program units.

- Syntax and semantics provide a language's definition

5.1.4 Previous Example

$$\underbrace{\text{Fed}}_{\text{verb}}\quad \underbrace{\text{I}}_{\text{subject}}\quad \underbrace{\text{the dog}}_{\text{object}}\quad \underbrace{.}_{\text{full stop}}$$

Both are syntactically and semantically not reasonable

$$\underbrace{\text{I}}_{\text{subject}}\quad \underbrace{\text{fed}}_{\text{verb}}\quad \underbrace{\text{the wall}}_{\text{object}}\quad \underbrace{.}_{\text{full stop}}$$

Syntactically reasonable but semantically not reasonable

$$\underbrace{\text{I}}_{\text{subject}}\quad \underbrace{\text{fed}}_{\text{verb}}\quad \underbrace{\text{the dog}}_{\text{object}}\quad \underbrace{.}_{\text{full stop}}$$

Both syntactically and semantically reasonable

5.1.5 Describing Syntax and Semantics (Example 1)

- Syntax is defined using some kind of rules
  - Specifying how statements, declarations, and other language constructs are written
- Semantics is more complex and harder to define
- Detecting syntax error is easier, semantics error is much harder

Example 1

if statement in C

Syntax: **if (<expr>) <statement>**

Semantics:

- if **<expr>** is evaluates to a nonzero number, execute **<statement>**
- Otherwise, control continues after the **if** construct

5.1.6 Describing Syntax and Semantics (Example 1)

Example 2

while statement in Java

Syntax: **while** (boolean_expr) statement

Semantics:

- When the current value of the Boolean expression is true, the embedded statement is executed
- Otherwise, control continues after the **while** construct

5.2.1 The General Problem of Describing Syntax-What is a Language

- In programming language terminologies
  - A language is a set of sentences
  - A sentence (or statement) is a strings of characters over some alphabet

Note:
A “sentence” is very general

In English, it may be an English sentence, paragraph, all text in a book, or hundreds of books

In programming: For example, if a C program can be compiled properly, is a sentence of the C language (no matter whether is a “hello world” or a program with several million lines of code)

5.2.2 A Sentence in C Language

- The “Hello World” program is a sentence in C

```
main()
{
   printf("hello, world!\n");
}
```

- What about its alphabet?

  -For illustration purpose, let us define the alphabet as

  a → identifier   b → string   c→ (
  d→)   e→ {   f→}   g→ ;

  - So, symbolically “Hello World” program can be represented by the sentence:
    acdeacbdgf
    where “main” and “printf” are identifiers and “hello, world!\n” is a string

Identifiers: name of variables, function name, class, arrays etc.

5.2.3 Sentence and Language

- So, we say that acdeacbdgf is a sentence of the C language, because it represents a legal program in C
  - Note: “legal” means syntactically correct

- How about the sentence acdeacbdf?
  - It represents the following program:

```
main()
{
        printf("hello, world!\n")
}
```

  - Compiler will say there is a syntax error
  - In essence, it says the sentence acdeacbdf is not in C language

5.2.4 Definition of a Language

- The syntax rules of a language specify which sentences are in the language, i.e., which sentences are legal sentences of the language

Example:

- Consider a special language X containing with syntax rule as

  A sentence consists of a noun followed by a verb, followed by a preposition, and followed by a noun,

  where a noun is a place

  a verb can be “is” or “belongs” and

  a preposition can be “in” or “to”

5.2.5 Syntax Rule Representation

- A more concise representation:

  <sentence> → <noun> <verb> <preposition> <noun>

  <noun> → *place*

  <verb> → “is” | “belongs”

  <preposition> → “in” | “to”

- With these rules, we can generate followings:

  Seoul is in Korea

  Korea is in Seoul

  Seoul belongs to Korea

- They are all in language X
  - Its alphabet includes “is”, “belongs”, “in”, “to”, *place*

5.2.6 Checking Syntax of a Sentence

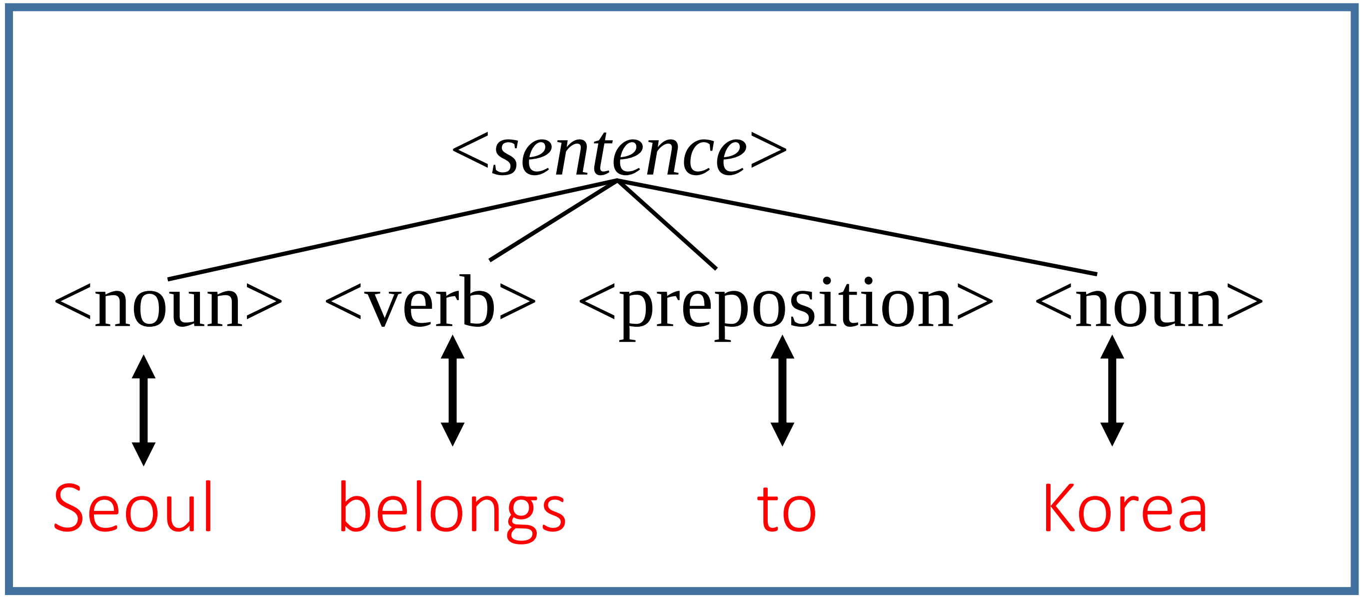
- How to check if the following sentence is in the language X?

  Seoul belongs to Korea

- Idea: Check if we can generate that sentence
  → This is called *parsing*

- How?
  Try to match the input sentence with the structure of the language

5.2.7 Matching the Language Structure



So, the sentence is in the language X!

The above structure is called a parse tree

5.3.1 Formal Description of Syntax

Most widely known methods for describing syntax:

- Context-Free Grammars
  - Developed by Noam Chomsky, a noted linguist, in the mid-1950s
  - Define a class of languages: context-free languages
  - Turned out to be useful for describing the syntax of programming languages

- Backus-Naur Form (1959)
  - Invented by John Backus to describe ALGOL 58
  - BNF is a natural notation for describing syntax
  - Equivalent to context-free grammars

5.3.2 Why BNF

- Before BNF, people specified programming languages ambiguously, i.e. with English

“He fed her cat food”

- Can have different meanings,

  - He fed a woman’s cat some food
  - He fed a women some food that was intended for cats

5.3.3 BNF Terminologies

- Lexeme
  - A lexeme is the lowest level syntactic unit of a language (E.g. Seoul, Korea, is, in etc. in previous example)
  - The lexemes of a programming language include its numeric literals, operators, and special words, among others.
  - Example of lexemes: sum, begin, +, *, etc.
  - Programs -> strings of lexemes rather than of characters

- Token
  - Token is a category of lexemes
  - Example of tokens: identifier (E.g. place in previous example)

5.3.4 Lexemes and Tokens (Example)

Q) Identify lexemes and corresponding tokens in the given java statement

index = 2 * count + 17;

Lexemes	Tokens
index	Identifier
=	equal_sign
2	int_literal
*	mult_op
count	identifier
+	plus_op
17	int_literal
;	semicolon

5.3.5 BNF Fundamentals-Revisiting Previous Example

Start (also a non-terminal)

```
<sentence> → <noun> <verb> <preposition><noun>
    <noun> → place
    <verb> →  “is”| “belongs”
<preposition> →  “in”| “to”
```

Production Rules

Nonterminal

Terminal

5.3.6 BNF Fundamentals

- Metalanguage for programming languages
- BNF formal method of defining a grammar with 4 tuples

1. A set of ***terminal*** symbols

    - Elementary symbols of the language defined by a formal grammar

    -Terminals are basically set of lexemes or tokens

    - Correspond to valid words in the vocabulary.

Metalanguage: A form of language or set of terms used for the description or analysis of another language

5.3.7 BNF Fundamentals

2. A set of ***nonterminal*** symbols

    - Replaced by groups of terminal symbols according to the production rules
    - enclosed in angle brackets i.e. < >
    - Correspond to legal phrases or sentences formed using words from the vocabulary.

3. A set of rules known as **productions** which can transform each non-terminal into a sequence of terminals.

    -a finite non-empty set of rules

4. The ***start*** symbol

    -special element of the nonterminals of a grammar

5.3.8 BNF Fundamentals

Note:

- Lexemes appear literally in program text
- Non-terminals stand for larger pieces of syntax
  - Do NOT occur literally in program text
  - The grammar says how they can be expanded into strings of tokens or lexemes
- The *start* symbol is the particular non-terminal that forms the starting point of generating a sentence of the language

5.3.9 BNF Rules

- A rule has
  - -a left-hand side (LHS) is a single non-terminal
  - -a right-hand side (RHS) contains one or more terminals or non-terminals

- A rule tells how LHS can be replaced by RHS, or how RHS is grouped together to form a larger syntactic unit (LHS)

General form of Production

LHS→ RHS

i.e.

Non-terminals →Terminals/non-terminals

5.3.10 BNF Rules

- Nonterminal symbols can have two or more distinct definitions
- Multiple definitions can be written as a single rule, with the different definitions separated by the symbol|, meaning logical OR

Example

```
<stmt> → <single_stmt>
              | begin <stmt_list> end
```

5.3.11 Describing Lists

- Variable- length lists in mathematics are often written using an ellipsis (. . .);

  E.g. 1, 2, ...
- BNF does not include the ellipsis, so an alternative method is required
- Syntactic lists are described using recursion

Example

```
<ident_list> → ident
             |  ident, <ident_list>
```

5.3.12 An Example Grammar

```
  <program> → begin <stmt_list> end
<stmt_list> → <stmt>
            | <stmt> ; <stmt_list>
     <stmt> → <var> = <expr>
      <var> → A | B | C |
     <expr> → <var> + < var >
            | < var > - < var >
            |<var>
```

<program> is the start symbol

a, b, c,+, -, ; , = are the terminals

5.3.13 Derivations

- A derivation is a repeated application of rules
  - starting with the start symbol, and
  - ending with a sentence (all terminal symbols)

- The start symbol is often named <program>

5.3.14 An Example Grammar and Derivation (Example)

Example : Grammar

```
<program> → <stmts>
 <stmts> → <stmt> | <stmt> ; <stmts>
 <stmt> → <var> = <expr>
 <var> → a | b | c | d
 <expr> → <term> + <term> | <term> - <term>
 <term> → <var> | const
```

Example :Derivation

```
<program> => <stmts> => <stmt>
                     => <var> = <expr>
                     => a = <expr>
                     => a = <term> + <term>
                     => a = <var> + <term>
                     => a = b + <term>
                     => a = b + const
```

5.3.15 Practice Question

Q) For the given grammar, derive `A=B*(A+C)`

```
<assign> → <id>=<expr>
<id> → A|B|C
<expr> → <id>+<expr>
        | <id>*<expr>
        | (<expr>)
        | <id>
```

5.3.16 Solution

```
<assign> → <id>=<expr>
<id> → A|B|C
<expr> → <id>+<expr>|<id>*<expr>|(<expr>)|<id>
```

Solution

```
Assign => <id>=<expr>
       => A=<expr>
       => A=<id>*<expr>
       => A=B*<expr>
       => A=B*(<expr>)
       => A=B*(<id>+<expr>)
       => A=B*(A+<expr>)
       => A=B*(A+<id>)
       => A=B*(A+C)
```

Q & A