
بسم الله الرحمن الرحيم

Introduction

Dr. Nasir Jalal PhD (Computer Science)

Lecturer

CS & IT Department, CUVAS Bahawalpur

Course Detail

BS CS 5th Semester Fall 2022-2026

Course: Theory of Programming Languages

Course Code: CS IT 507

Lecture: 2 (12-09-2024)

Outline Lecture 2

- Revision of Lecture 1
- Reasons for Studying Concepts of Programming Languages
- Error and its types
- Characteristics of Programming Language
- Some Common Characteristics of Programming Language

Theory of Programming Languages

The **theory of programming languages** is a branch of computer science that focuses on the design, implementation, analysis, and classification of programming languages. It deals with understanding the fundamental principles behind how languages function and how they can be used to solve problems efficiently.

Reasons for Studying Concepts of Programming Languages

- Increased capacity to express ideas.
- Improved background for choosing appropriate languages
- Increased ability to learn new languages.
- Better understanding of the significance of implementation.
- Better use of languages that are already known.
- Overall advancement of computing.

Characteristics of Programming Language

There are three characteristics of each language

- Syntax
- Semantics
- Pragmatics

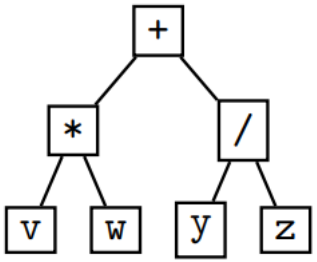
Syntax

- It refers to the rules and regulations for writing any statement in a programming language
- A statement is syntactically valid if it follows all the rules.
- It is related to the grammar and structure of the language.

Syntax

Consider a phrase that indicates the sum of the product of v and w and the quotient of y and z. The different for of the phrase are

- vw + y/z
- (+ (*v w) (/yz))
- In form of tree



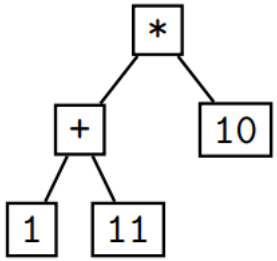
The syntax of a programming language specifies which concrete notations (above three) in the language are legal

Semantics

- It refers to the meaning associated with the statement in a programming language.
- Errors are handled at runtime.
- Semantics may be consider as logic.

Semantics Example

- Consider the following tree structure



- for usual Decimal Notations Result is (1+11).10 = 120
- if * is consider for Exponent Result is 12^{10}
- If the numerals are in binary notation Result will be $(1 + 3) \cdot 2 = 8$

Semantics Example

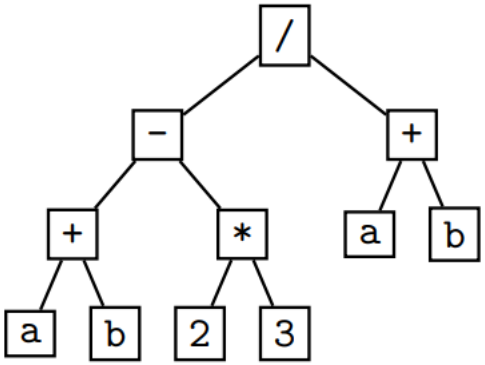
This example illustrates how a single program phrase can have many possible meanings. Semantics describes the relationship between the abstract structure of a phrase and its meaning.

Pragmatics

Semantics deals with what a phrase means, pragmatics focuses on the details of how that meaning is computed. Of particular interest is the effective use of various resources, such as time, space, and access to shared physical devices (storage devices, network connections, video monitors, printers, speakers, etc.).

Pragmatics Example

As a simple example of pragmatics, consider the evaluation of the following expression tree (under the first semantic interpretation described above):



Suppose that `a` and `b` stand for particular numeric values. Because the phrase `(+ a b)` appears twice, a naive evaluation strategy will compute the same sum twice. An alternative strategy is to compute the sum once, save the result, and use the saved result the next time the phrase is encountered. The alternative strategy does not change the meaning of the program, but does change its use of resources; it reduces the number of additions performed, but may require extra storage for the saved result. Is the alternative strategy better? The answer depends on the details of the evaluation model and the relative importance of time and space.

Syntax, Semantics and Pragmatics

- Syntax — the form of programming languages.
- Semantics — the meaning of programming languages.
- Pragmatics — the implementation of programming languages

Some Common Characteristics

1. Simplicity
2. Abstraction
3. Efficiency
4. Portability
5. Readability
6. Maintainability
7. Extensibility
8. Flexibility
9. Type System
10. Concurrency
11. Error Handling
12. Modularity
13. Security
14. Reusability
15. Interoperability
16. Object-Oriented Features
17. Functional Programming Features
18. Expressiveness
19. Support for Libraries and Frameworks
20. Paradigm Support

THANKS