C SC 520: Principles of Programming Languages

Peter J. Downey

Department of Computer Science

Spring 2006

Principles of Programming Languages

Lecture 01

Introduction

A Programming Language

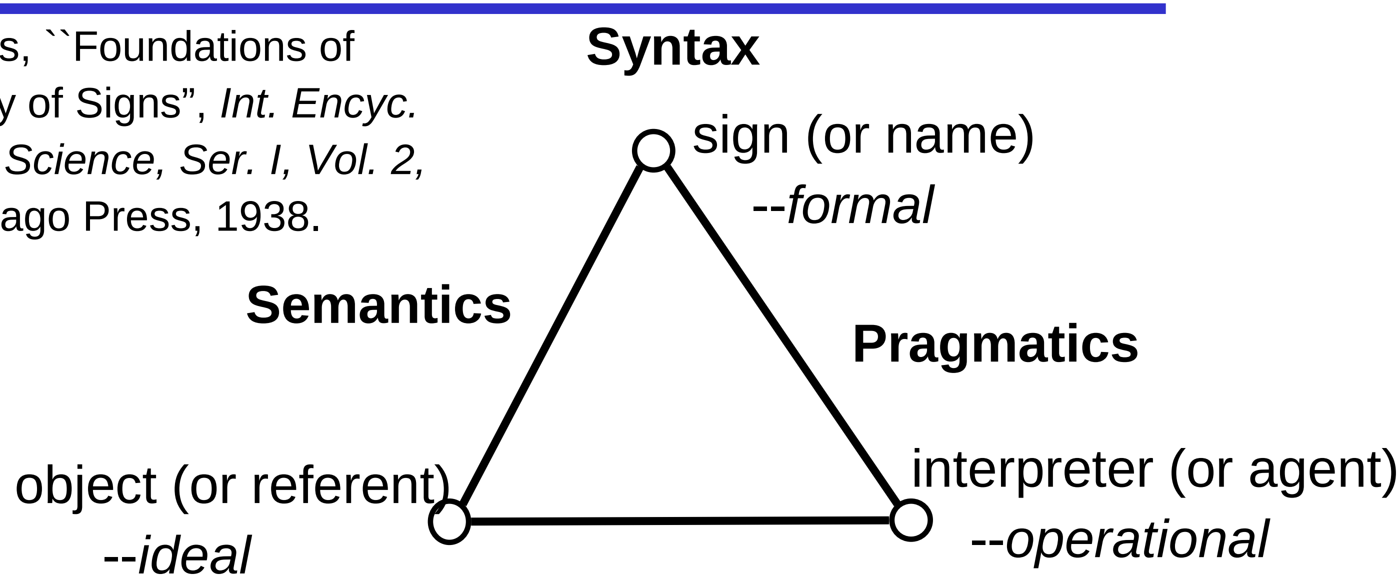
- Notation for describing algorithms and data structures
- Medium for communicating procedural actions to an interpreting agent (machine or man)
- Mental tool for
  - Solving procedural problems
  - Representing algorithms
  - Reasoning about algorithms
- Specification of a *virtual* computer

Reasons to study programming languages

- To understand the connection between languages and the problem solving process—how it conditions our thinking
- To suggest designs for languages suited to needs of problem solving and software production—the isolation of universals
- To permit a better choice of programming language for a particular problem
- To understand the meaning of one language by comparison with others—development of semantic description tools
- To understand how languages and features are implemented
- To make it easier to learn new languages

Semiotics: the study of *signs* and *systems of signs*

--C.W. Morris, ``Foundations of the Theory of Signs", *Int. Encyc. of Unified Science, Ser. I, Vol. 2,* U of Chicago Press, 1938.



- *Syntactics:* relations between signs (in abstraction from their associations with objects or interpreters)
- *Semantics:* relations between signs and objects they denote; the study of sign meaning, including relations among objects denoted
- *Pragmatics:* nature of sign-interpreters and the origin, uses and effects of signs on interpreters

Ex: The Language *Binary Numerals*

- *Syntax:* signs
  - Numerals (syntactic category **<num>**)
  - Abstract syntax:

    `<num>  ::=  <num> 0 | <num> 1 | 0 | 1`

- *Semantics:* sign → object
  - A mapping (semantic map) M from **numerals** to **integers:**
    - $M[\![\, \mathbf{101}]\!] = 5 \qquad M[\![\mathbf{000101}]\!] = 5$

    $M : \mathbf{< num >} \rightarrow \mathbf{Integer}$
  - Defined by ''syntax-driven" (compositional) semantics
    - "meta-variable N ranges over elements of the syntactic category **<num>**

Binary Numerals (cont.)

$$\mathcal{M}[\![\mathbf{0}]\!] = 0$$
$$\mathcal{M}[\![\mathbf{1}]\!] = 1$$
$$\mathcal{M}[\![N\mathbf{0}]\!] = 2 \cdot \mathcal{M}[\![N]\!]$$
$$\mathcal{M}[\![N\mathbf{1}]\!] = 2 \cdot \mathcal{M}[\![N]\!] + 1$$

- Another semantic notion is ``semantic equivalence'' $\equiv$

$$\mathbf{101} \equiv \mathbf{0101} \text{ since } \mathcal{M}[\![\mathbf{101}]\!] = \mathcal{M}[\![\mathbf{0101}]\!]$$

- *Pragmatics:* interpreter $\rightarrow$ object
  - Design of an interpreter to check $\equiv$
  - Algorithm *Add* to perform *semantically valid* addition of symbols

$$\mathcal{M}[\![Add(N_1, N_2)]\!] = \mathcal{M}[\![N_1]\!] + \mathcal{M}[\![N_2]\!]$$

Ex: C

- *Syntax:*
  - C grammar
  - C parser
- *Semantics:*
  - Axiomatic semantic specification

    $\{Q[\mathbf{e} / \mathbf{x}]\}$ **x = e** $\{Q\}$

    $\{2y - 3 > 25\}$ **x = 2 * y - 3** $\{x > 25\}$ or simplified: $\{y > 14\}$ **x = 2 * y - 3** $\{x > 25\}$
  - Denotational specification using syntax-directed (compositional) rules $M([\![\mathbf{x = e}]\!], env, mem) = \text{update location}$

    $find(env, \mathbf{x}) \text{ with value } E([\![\mathbf{e}]\!], env, mem)$
- *Pragmatics:*
  - Implementation techniques
  - Programming methodology given C’s features

Reasons for Semantic Description

- Main aim: each phrase of language is given a denotation (meaning, referent) determined only by the meaning of its subphrases
- Benefits
  - Standard of definition
  - Basis for design comparisons
  - Basis for correctness, validation
  - Provides insight
- Methods
  - Informal semantics (e.g., Algol 60): incomplete, even inconsistent
  - Operational semantics (e.g., standard implementation): it is what it does—meaning and pragmatics confused
  - Axiomatic semantics: meaning of phrase is a *predicate transformation.* Directly supports verification
  - Denotational semantics: every phrase denotes a *thing* (integer, boolean, mathematical function)