Introduction
Description of programming languages

• Syntax –

The structure and shape of what can be written in the language.

• Semantics –

The meaning of all what can be written in the language. Also extracts the meaningful syntactic objects from meaningless ones.

• Pragmatics –

The usability matters of the language, expected applications, purposes, implementations etc.
Notions about syntax

• Syntactic category –

A set of all syntactic objects of a language with similar usage.

Typical syntactic categories: literals, variables, expressions (entailing literals and variables), statements, procedures.

• Abstract syntax specifies the syntactic categories of the language and the constructs for building larger syntactic objects out of smaller ones, in particular the number and syntactic categories of their constituents and the syntactic category of the resulting syntactic objects.

*Semantics is defined for abstract syntax. Concrete notation doesn’t matter.*
Classification of semantics

• Big vs small step semantics:
  
  A big step semantics describes the whole computation at once;
  
  A small step semantics describes elementary execution steps.

• Dynamic vs static semantics:
  
  A dynamic semantics describes execution of programs;
  
  A static semantics describes properties independent of execution.

• Standard vs non-standard semantics:
  
  A standard semantics corresponds to one’s intuition about meaning;
  
  A non-standard semantics deviates from the intuition.
Applications of semantics

• Establishing correctness of programs.
• Establishing correctness of interpreters and compilers.
• Program analysis.
Syntax specification facilities

• Concrete syntax is classically described by context free grammars (CFG).

• Abstract syntax is often described by similar means.

• Semantics is “everything that can’t be represented by CFG”.
A toy language While

• Abstract syntax of While (blackboard):
  – category of variables as collections of atomic units, decimal numerals, arithmetic expressions, boolean expressions, statements as in the book;
  – expressions of distinct types being in distinct categories is untypical.

• Abstract syntax trees show the structure of code (blackboard):
  – Example in Fig. 1.1;
  – Exercise 1.1.

• Derivation trees show the categories of constituents together with connectives (blackboard): the same example and exercise.
Semantics description methods

• Operational semantics specifies the meaning of code as a set of computation steps via derivation rules and axioms.

• Denotational semantics specifies the meaning of code in terms of mathematical objects, co-called denotations. The correspondence between syntactic and semantic objects is carried by semantic functions, defined separately for each syntactic category.

• Axiomatic semantics specifies the meaning of code as effects of executing the code on assertions, again via derivation rules and axioms.
Denotational semantics of numerals

• Semantic function (blackboard):
  – Type;
  – Definition;
  – Structural induction and compositionality.

• Example 1.3 for decimal numerals (blackboard).

• Fact 1.5 for decimal numerals (blackboard).

• Exercise 1.4 for decimal numerals (blackboard).
Denotational semantics of variables

• States (blackboard).

• Semantic function (blackboard):
  – Type;
  – Definition.
Denotational semantics of arithmetic expressions

• Semantic function (blackboard):
  – Type;
  – Definition (Table 1.1);
  – Structural induction and compositionality.

• Example 1.6 (blackboard).
Denotational semantics of boolean expressions

• Semantic function (blackboard):
  – Type;
  – Definition (Table 1.2);
  – Structural induction and compositionality.

• Exercise 1.9 (oneself).
Free variables

- Definition (blackboard).
- Lemma 1.12 (blackboard).
- Exercise 1.13 (blackboard).
Substitutions and updates

• Definitions (blackboard):
  – Substitution;
  – Update.

• Exercise 1.14 together with an improvised example (blackboard).

• Exercise 1.15 (home).