Denotational Semantics

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Programming Language

Any notation for giving instructions, e.g. Pascal, input commands to application program, ...

- Syntax
  - Appearance and structure of sentences.
- Semantics
  - Assignment of meaning to sentences e.g. numbers, functions, machine actions, ...
- Pragmatics
  - Usability of the language e.g. application areas, performance, ...

Features of every computer program.

Implementation

Two main parts:

1. Input checker module ("Parser")
   - Reads input, verifies that it has proper syntax, generates internal representation.
2. Evaluation module
   - Evaluates input to corresponding output thus defining the semantics of the language.

The implementation of language is a pragmatic issue.

Evaluation

- Interpretation
  - Execution of the program.

  Interpreter defines meaning (by its actions).

- Compilation
  - Transformation of the program into an equivalent version in the machine language.

  Compiler preserves meaning (by notion of "equivalence").

Formal Specifications

- Syntax: Backus-Naur Form (BNF)
  - Correspondence between BNF and parser.
  - Input to parser generator.
- Semantics: ?
  - Precise Standard for implementation.
  - Useful user documentation.
  - Tool for design and analysis of language
  - Input to compiler generator.

Semantics is much more difficult to describe ("semantics is everything that cannot be described in BNF").

Operational Semantics

- Language defined by interpreter (abstract machine).
- Each construct defined by a transition rule.
- Meaning of a program is a sequence of interpreter states.

$$P : S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$$

- P ... program
- S_i ... interpreter state

Notation for interpreter may be as complex as language itself.

Axiomatic semantics

- Language defined by system of logical axioms and inference rules.
- Each construct defined by an axiom.
- Not the meaning of a program but its properties are defined.

$$\{A\}\ P\ \{B\}$$

P ... program
A ... precondition
B ... postcondition

Provable properties need not characterize program uniquely.

Denotational semantics

- The meaning of a program is a (mathematical) object.
- Each construct is mapped by a valuation function into its meaning (denotation).

$$F(P) = D$$

P ... program
F ... valuation function
D ... denotation

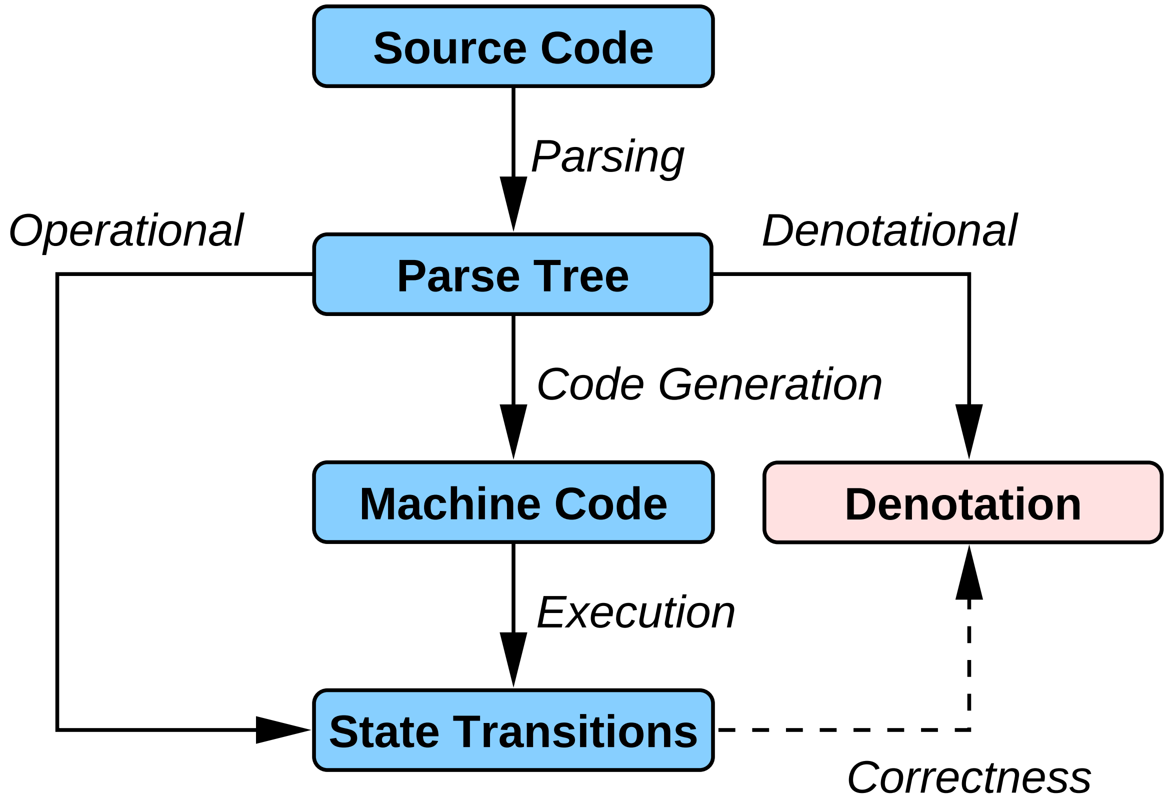
More abstract than operational semantics (no computation steps), more concrete than axiomatic semantics (explicit meaning).

Application Areas

- Axiomatic: initial specification.
  - Which properties shall language have?
- Denotational: meaning.
  - Which semantics provides properties?
- Operational: implementation.
  - How can semantics be implemented?

Complementary aspects.

Relationship



Correctness of implementation can be verified with respect to the denotation.

Valuation Function

- Domain: Abstract syntax structures ("parse trees") of the language.
- Target: Objects of semantic domains.
- Structural definition (meaning of a tree is defined by meanings of its subtrees).

A valuation function maps an abstract syntax into some semantic domain.

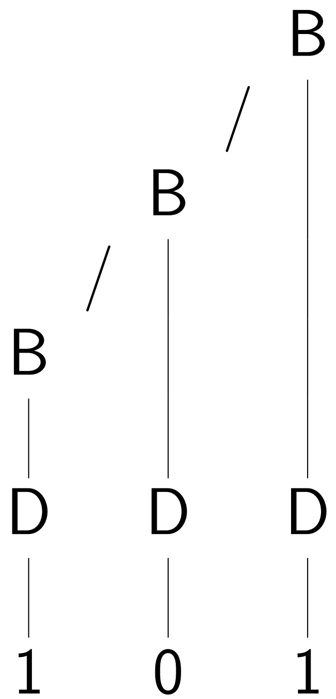
A Language of Binary Numerals

- Abstract Syntax:

  Syntactic domains:
  B ∈ Binary-numeral
  D ∈ Binary-digit

  Syntax rules:
  B ::= BD | D
  D :: = 0 | 1

- Sentences:



Meaning of Terminal Sentences

- Subtree: $\begin{array}{c} \mathsf{D} \\ | \\ 0 \end{array}$

- Meaning: $\mathsf{D}\left(\begin{array}{c} \mathsf{D} \\ | \\ 0 \end{array} \right) = \mathit{zero}$

- Notation: $\mathsf{D}\,[[\,0\,]] = \mathit{zero}$

  Valuation function:
  $\mathsf{D}\,[[\,0\,]] = \mathit{zero}$
  $\mathsf{D}\,[[\,1\,]] = \mathit{one}$

Meaning of Non-Terminal Sentences

- Subtree: $\begin{array}{c} \mathsf{B} \\ | \\ \mathsf{D} \\ | \\ 1 \end{array}$
- Meaning: $\mathsf{B}\left(\begin{array}{c} \mathsf{B} \\ | \\ \mathsf{D} \\ \triangle \end{array} \right) = \mathsf{D}\left(\begin{array}{c} \mathsf{D} \\ \triangle \end{array} \right)$
- Notation: $\mathsf{B}[\![\ \mathsf{D}\ ]\!] = \mathsf{D}[\![\ \mathsf{D}\ ]\!]$

  Valuation function:

  $\mathbf{B}[\![\ \mathsf{D}\ ]\!] = \mathbf{D}[\![\ \mathsf{D}\ ]\!]$

  $\mathbf{B}[\![\ \mathsf{BD}\ ]\!] =$

  $(\mathbf{B}[\![\ \mathsf{B}\ ]\!]\ \textit{times two})\ \textit{plus}\ \mathbf{D}[\![\ \mathsf{D}\ ]\!]$

Meaning of Non-Terminal Sentences

Abstract Syntax

B $\in$ Binary-numeral
D $\in$ Binary-digit
B ::= BD | D
D :: = 0 | 1

Semantic Algebras

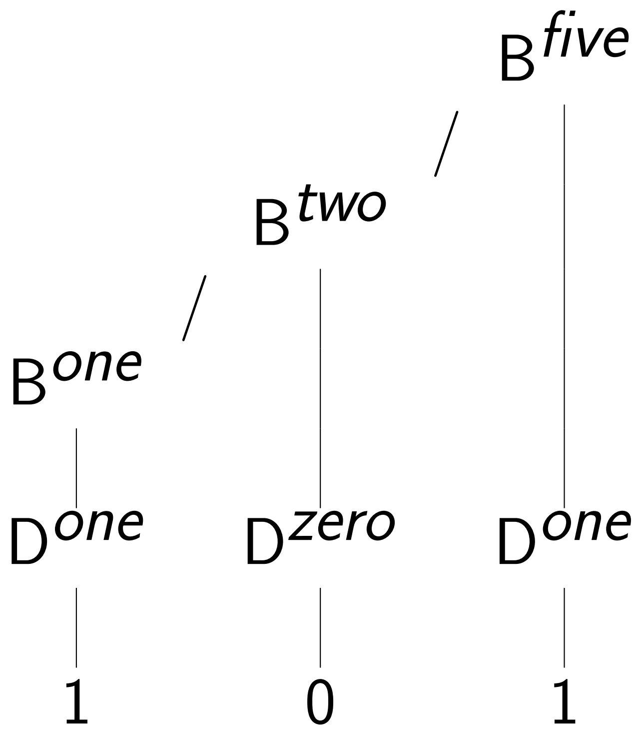
Natural Numbers:
Domain $Nat = \mathbf{N}$
Operations *zero*, *one*, *two*, . . . : *Nat*
plus, *times*: $Nat \times Nat \rightarrow Nat$

Valuation Functions

B: Binary-numeral $\rightarrow$ *Nat*
B[[D]] = **D**[[D]]
B[[BD]] = (**B**[[B]] *times two*) *plus* **D**[[D]]
D: Binary-digit $\rightarrow$ *Nat*
D [[0]] = *zero*
D [[1]] = *one*

Meaning of Sentence

Annotation of abstract syntax tree



Computation can proceed bottom-up or top-down!