


Sebesta, Concepts of Programming Languages, 4th Ed

Chapter 3:  
Describing Syntax and Semantics



Oregon State University  
Open mind. Open doors.

---

---

---

---

---


---

---

Syntax and Semantics

- **Syntax** - the form or structure of the expressions, statements, and program units
- **Semantics** - the meaning of the expressions, statements, and program units

CS 381 Prog Lang Chapter 3 2



Oregon State University  
Open mind. Open doors.

---

---

---

---


---

---

---

Who Uses Language Definitions?

- 1. language designers
- 2. Implementors
- 3. Programmers (the users of the language)



Oregon State University  
Open mind. Open doors.

---

---

---

---

---

---

---

Some Vocabulary

- A *sentence* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., \*, sum, begin)
- A *token* is a category of lexemes (e.g., identifier)

OSAKA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 4

---

---

---

---

---

---

---

---

Context-Free Grammars

- Developed by Noam Chomsky in the mid-1950s
- - Language generators, meant to describe the syntax of natural languages
- - Define a class of languages called *context-free languages*

OSAKA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 5

---

---

---

---

---

---

---

---

Backus Naur Form

- Invented by John Backus and Peter Naur to describe Algol 58
- BNF is equivalent to context-free grammars
- A *metalanguage* is a language used to describe another language.

OSAKA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 6

---

---

---

---

---


---

---

---

Rules and Nonterminals

- In BNF, *abstractions* are used to represent classes of syntactic structures—they act like syntactic variables (also called *nonterminal symbols*)
- e.g.
- `<while_stmt> -> while <logic_expr> do <stmt>`
- This is a *rule*; it describes the structure of a while statement


CS 381 Prog Lang Chapter 3 7

---

---

---

---

---


---

---

---

Grammars and rules

- A rule has a left-hand side (LHS) and a right-hand side (RHS), and consists of *terminal* and *nonterminal* symbols
- A *grammar* is a finite nonempty set of rules
- A *derivation* is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)


CS 381 Prog Lang Chapter 3 8

---

---

---

---

---


---

---

---

Right Hand Sides

- An abstraction (or nonterminal symbol) can have more than one RHS
- `<stmt> -> <single_stmt>`  
 • `| begin <stmt_list> end`
- Syntactic lists are described in BNF using recursion
- `<ident_list> -> ident`  
 • `| ident, <ident_list>`


CS 381 Prog Lang Chapter 3 9

---

---

---

---

---

---

---

---

### An Example Grammar

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

OSAKAWA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 10

---

---

---

---

---

---

---

---

### An Example Derivation

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

OSAKAWA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 11

---

---

---

---

---

---

---

---

### More Vocabulary

- Every string of symbols in the derivation is a *sentential form*
- A *sentence* is a sentential form that has only terminal symbols
- A *leftmost derivation* is one in which the leftmost nonterminal in each sentential form is the one that is expanded
- A derivation may be neither leftmost nor rightmost

OSAKAWA UNIVERSITY  
Open mind. Open heart.

CS 381 Prog Lang Chapter 3 12

---

---

---

---

---


---

---

---

### Parse Trees

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


CS 381 Prog Lang Chapter 3 13

---

---

---

---

---

---


---

---

### Ambiguity

•A grammar is *ambiguous* iff it generates a sentential form that has two or more distinct parse trees

An ambiguous expression grammar:  
`<expr> -> <expr> <op> <expr> | const`  
`<op> -> / | -`


CS 381 Prog Lang Chapter 3 14

---

---

---

---

---


---

---

---

### Remove Ambiguity by Adding Nonterminals

An unambiguous expression grammar:  
`<expr> -> <expr> - <term> | <term>`  
`<term> -> <term> / const | const`


CS 381 Prog Lang Chapter 3 15

---

---

---

---

---

---

---

---

Associativity, Precedence can be Specified by Grammars

```

<expr> => <expr> * <term> | <term>
<term> => <term> + <factor> | <factor>
<factor> => const

```

OSU  
OHIO STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 16

---

---

---

---

---

---

---

---

Extended BNF

- Extended BNF (just abbreviations):
- 1. Optional parts are placed in brackets ([])
  - <proc\_call> -> ident [ ( <expr\_list> ) ]
- 2. Put alternative parts of RHSs in parentheses and separate them with vertical bars
  - <term> -> <term> ( + | - ) const
- 3. Put repetitions (0 or more) in braces ({} )
  - <ident> -> letter { letter | digit }

OSU  
OHIO STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 17

---

---

---

---

---

---

---

---

EBNF makes grammars smaller

- BNF:
  - <expr> -> <expr> + <term>
  - | <expr> - <term>
  - | <term>
  - <term> -> <term> \* <factor>
  - | <term> / <factor>
  - | <factor>
- EBNF:
  - <expr> -> <term> { ( + | - ) <term> }
  - <term> -> <factor> { ( \* | / ) <factor> }

OSU  
OHIO STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 18

---

---

---

---

---


---

---

---

Syntax Graphs

put the terminals in circles or ellipses and put the nonterminals in rectangles; connect with lines with arrowheads



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 19

---

---

---

---

---

---


---

---

Semantics

Unlike Syntax, there is no one widely accepted notation or formalism for describing semantics. Three most widely used are

- Operational Semantics
- Axiomatic Semantics
- Denotation Semantics



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 20

---

---

---

---

---


---

---

---

Operational Semantics

- Describe the meaning of a program by executing its statements on a machine, either simulated or actual. The change in the state of the machine (memory, registers, etc.) defines the meaning of the statement
- That is, we define a language by a (virtual or real) machine.



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 21

---

---

---

---

---

---


---

---

Axiomatic Semantics

**Based on Formal Logic (Predicate Calculus)**

**Approach: Define axioms and inference rules for each statement in language.**



CS 381 Prog Lang Chapter 3 22

---

---

---

---

---

---

---

---


An Inference Rule for Sequential Execution

**For a sequence S1;S2:**

$\{P1\} S1 \{P2\}$   
 $\{P2\} S2 \{P3\}$

**the inference rule is:**

$\{P1\} S1 \{P2\}, \{P2\} S2 \{P3\}$   
 $\{P1\} S1; S2 \{P3\}$



CS 381 Prog Lang Chapter 3 23

---

---

---

---

---

---

---


---

An Inference Rule for Loops

**•For the loop construct:**

- $\{P\} \text{ while } B \text{ do } S \text{ end } \{Q\}$
- the inference rule is:**
  - $(I \text{ and } B) S \{I\}$
  - $\{I\} \text{ while } B \text{ do } S \{I \text{ and } (\text{not } B)\}$

**•where I is the loop invariant.**



CS 381 Prog Lang Chapter 3 24

---

---

---

---

---


---

---

---

Evaluation of Axiomatic Semantics

1. Developing axioms or inference rules for all of the statements in a language is difficult
2. It is a good tool for correctness proofs, and an excellent framework for reasoning about programs, but it is not as useful for language users and compiler writers



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 25

---

---

---

---

---

---

---


---

Denotational Semantics

The most abstract semantics technique, based on recursive function theory

The process of building a denotational spec for a language:

1. Define a mathematical object for each language entity
2. Define a function that maps instances of the language entities onto instances of the corresponding mathematical objects



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 26

---

---

---

---

---


---

---

---

Evaluation of Denotational Semantics

- Can be used to prove the correctness of programs
- Provides a rigorous way to think about programs
- Can be an aid to language design
- Has been used in compiler generation systems



OSU  
OREGON STATE UNIVERSITY  
Open mind. Open doors.

CS 381 Prog Lang Chapter 3 27

---

---

---

---

---

---

---

---