Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 12, Part 1: Overview of Semantics

© 2007 Jones and Bartlett Publishers

Programming Language Semantics

- A programming language is characterized by syntax and semantics.
- The syntax of the language indicates the exact form and structure of elements that may be produced or recognized (parsed),
- The formal semantics of a programming language relate the language syntax to meaning, which is in many cases a quantitative description of the underlying computation. There are a number of different approaches to formally quantifying semantics.

© 2007 Jones and Bartlett Publishers

Combining syntax and semantics leads to the oft-cited result:

Programming Language = Syntax + Semantics

© 2007 Jones and Bartlett Publishers

Objectives

- Formal approaches to semantics have the goal of **achieving a precise, mathematical characterization of the behavior of an assumed syntactically correct program in a specific language**.
- A standardized and generally-accepted mathematical formalism for this purpose has been an elusive (and sometimes controversial) goal.

© 2007 Jones and Bartlett Publishers

Given the following syntactically correct `minic` (or `c`) program:

```
int main(void)
{
int t1;
t1=10;
return(t1);
}
```

The basic question is:

> What is the **meaning** of this program?

To answer this, we must formally and quantitatively define *meaning*.

© 2007 Jones and Bartlett Publishers

The Road Map, Updated

concept	process	resulting mapping
lexical analysis	scanning	input file → Tokens
syntactic analysis	parsing	Tokens → abstract parse tree
semantic analysis	parse interpretation	abstract parse tree → semantic objects

Table 1: Enhanced Interpretation Processes for a Programming Language

© 2007 Jones and Bartlett Publishers

Semantics, Semantics, Semantics

Our overall motivation is to:

- Describe what a program does or produces (usually by description of *components* of the program)
- Describe what happens during the execution of a program (or program component)
- Describe *meaning* of statements in a programming language
- Describe what a syntactic fragment should produce via machine translation.

© 2007 Jones and Bartlett Publishers

Typically, any approach to studying semantics has the following characteristics:

- Usually a 'divide and conquer', or **compositional** approach, is taken. We decompose a program into syntactic/semantic fragments and consider these individually. Therefore, we usually consider the tightly coupled syntax $\rightarrow$ semantics descriptions of these fragments.
- Also, typically an 'abstract' syntax is employed for simplicity and clarity.

© 2007 Jones and Bartlett Publishers

The utility of any formal approach to semantics varies with particular approach and overall objective(s) of the semantic formalization. While the approaches we present are 'formal' and usually quantitative, in some sense, they are also:

- (Usually) in a form which allows manipulation,
- Not one approach, but family of strategies, and
- Not fully accepted, i.e., possibly controversial.

© 2007 Jones and Bartlett Publishers

World's Simplest Example

Consider the following semantic representation example:

> *Describe the semantics of a syntactically correct arithmetic statement program fragment as the value of the statement.*

Consider the statement fragment[a]:

$$1101$$

[a]Which might be part of a more complex (and complete) assignment statement such as

$$E = V1 + 1101;$$

© 2007 Jones and Bartlett Publishers

This fragment could be

1. Interpreted syntactically as a string of digits (assuming it is syntactically correct); and
2. Interpreted or describe semantically as a *value*, i.e., the semantics or meaning could be 1101_{10} or 1101_2.

© 2007 Jones and Bartlett Publishers

- This semantic interpretation (evaluation) still requires we understand the positional notation used in the syntactic specification.
- To formalize the semantic specification (in this case determining value), we could develop a function `evaluate`, i.e.,

  `evaluate`$[1101] =$

  `evaluate`$[1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0] =$

  1101
- Unfortunately, the simplicity of this 'value' example obscures some difficult questions in formalizing semantics:
  - What about the semantics of control or print statements?
  - Whis is the role of the machine environment?

© 2007 Jones and Bartlett Publishers

Informal Operational Semantics

- An approach based upon specifying the operation of code fragments on an assumed machine.
- Typically used in 'language references' , 'how to program in xxx' and 'introduction to xxx' type programming language references and documentation.
- The way most programming is (initially) taught/learned.

© 2007 Jones and Bartlett Publishers

An example in `c` or `c++`.

In this form, the semantic description is often preceded by a quasi-formal syntactic description or prototype such as:

```
if (<condition>) <statement>
if (<condition>) <statement> else <statement>
```

This is then followed with an (informal) semantic description, for example:

> *In an* `if` *statement, the first (or only) statement is executed if the expression is nonzero and the second statement (if it is specified) is executed otherwise. This implies...*

© 2007 Jones and Bartlett Publishers

The Plan for the Study of Semantics

In what follows, we first give an overview of a number of differnt approaches. Then, several approaches are considered in detail.

© 2007 Jones and Bartlett Publishers

Semantics via Self-definition

An operational self-definition of the language is given by defining the interpretation of the language *in the language itself*, e.g., by showing an implementation of:

- A Lisp interpreter written in Lisp,
- A Prolog interpreter in written in Prolog, or
- A `c` interpreter written in `c`.

To some, this appears to be circular reasoning.

© 2007 Jones and Bartlett Publishers

Translational Semantics Overview

- Semantics are conveyed by showing the translation of program fragments (syntactically correct statements) into another (e.g., assembly language) language called the target language.
- This requires an actual or hypothetical machine (model), i.e., the target machine.
- Another viewpoint of translational semantics is that the semantics of the language are conveyed by defining an interpreter or compiler for the language.
- Translational semantics is a significant approach for compiler writers; but is of limited general utility in learning or expressing semantics.

© 2007 Jones and Bartlett Publishers

A `minic` Example of Translational Semantics. To illustrate translational semantics, we return to the simple `minic` program:

```
int main(void)
{
int t1;
t1=10;
return(t1);
}
```

Using `gcc`, the result (semantics) is the assembly code shown on the next slide.

© 2007 Jones and Bartlett Publishers

```
        .file   "minicex1.c"
        .version        "01.01"
gcc2_compiled.:
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $4, %esp
        movl    $10, -4(%ebp)
        movl    -4(%ebp), %eax
        movl    %eax, %eax
        leave
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident    "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)"
```

© 2007 Jones and Bartlett Publishers

Operational Semantics Overview

- Operational semantics convey language semantics by defining *how a computation is performed*, i.e., by describing the **actions** of the program fragment in terms of an actual or hypothetical machine.
- Operational semantics is also referred to as *intensional semantics*, because the sequence of internal computation steps (the intension) is most important.
- *informal* operational semantics is the typical means of describing programming languages.

© 2007 Jones and Bartlett Publishers

It is noteworthy that in operational semantics:

- An actual or hypothetical machine (model), i.e., the target machine is required.
- A **precise** description of the machine is required.
- The machines used tend to be simple.

© 2007 Jones and Bartlett Publishers

Operational Semantics Example

Suppose the state of the machine consists of the values of all registers, memory locations and condition codes and status registers.

We show the semantics with the following steps:

1. Record the machine state prior to execution/evaluation of the code fragment.
2. Execute/evaluate the code fragment (assume termination).
3. Examine the new machine state.

Thus, the semantics of the code fragment are conveyed by the change in the machine state.

© 2007 Jones and Bartlett Publishers

Denotational Semantics Overview

- *The semantics of an assumed syntactically correct program are represented by a mathematical mapping.*
- The steps taken to calculate the output are unimportant; it is the functional relationship of input to output that matters.
- Denotational semantics is also called *extensional semantics*, because the 'extension' or relation between input and output is the focus.
- Specifically, denotational semantics is used to map syntactic objects into domains of mathematical objects, i.e., an overall function is postulated with:

$$\texttt{meaning} : \texttt{Syntax} \rightarrow \texttt{Semantics}$$

© 2007 Jones and Bartlett Publishers

- The divide-and-conquer approach is important in denotational semantics, in the sense that the overall functional mapping is subdivided into a composition of functions and each of the composite functions is related to a program fragment.

© 2007 Jones and Bartlett Publishers

Simple Denotational Semantics Example

Recall our previous simplistic example, i.e., the semantic interpretation of the string 1101, i.e., the value 1101. Denotational semantics approaches this by defining a semantic function, D, with the corresponding mapping as follows:

$$D : digit|digit - string \rightarrow integer\ value$$

Notice the domain of this function is a syntactic construct and the range is a semantic concept.

© 2007 Jones and Bartlett Publishers

Axiomatic Semantics Overview

- Axiomatic semantics employ logic to convey meaning.
- The meaning of a syntactically correct program is formalized as a logical proposition or assertion (sometimes used as a program specification) that constrains the mapping between program input and output.
- The program semantics are based on assertions about logical relationships that remain the same each time the program executes.
- The basic representational strategy is:

$$\{logical\ preconditions\} \cap \{syntactic\ fragment\} \rightarrow$$

$$\{logical\ postconditions\}$$

© 2007 Jones and Bartlett Publishers

- Axiomatic semantics are more abstract than denotational semantics, and involve assertions and logical formulas from the predicate logic.
- Axiomatic semantics are **machine independent**.

© 2007 Jones and Bartlett Publishers

Algebraic Semantics Overview

In algebraic semantics, an algebraic specification of data and language constructs is developed. This approach is noteworthy, since it leads to the notion of abstract data types, which in turn could be thought of as implying OO-programming.

© 2007 Jones and Bartlett Publishers

Semantic Equivalence

Two different program fragments may have exactly the same meaning or semantics.

For example, consider the two `minic` fragments:

```
t1 = 5 + 4;
```

and

```
t1= 27 / 3;
```

© 2007 Jones and Bartlett Publishers

Notice the similarity between constructs such as

```
for(;;)
```

and

```
while(true)
```

This raises the often important issue of *semantic equivalence*.

The exact definition of semantic equivalence depends upon the specific semantic formalism used.

© 2007 Jones and Bartlett Publishers