Concepts of Programming Languages

Lecture 6 - Semantics

Patrick Donnelly

Montana State University

Spring 2014

Administrivia

Assignment:

Programming #1 : due 02.10
Homework #2 : due 02.19

Reading:

Chapter 3

Ishmael: Surely all this is not without meaning.

Moby Dick by Herman Melville

Thinking about Semantics

Definition

Semantics are the meaning of the expressions, statements, and program units.

Syntax and semantics together provide a language's definition.

Who uses these definitions?

- Other language designers
- Implementers
- Programmers (the users of the language)
- Standards developers

Semantics

There is no single widely acceptable notation or formalism for describing semantics.

Several needs for a methodology and notation for semantics:

- Programmers need to know what statements mean
- Compiler writers must know exactly what language constructs do
- Correctness proofs would be possible
- Compiler generators would be possible
- Designers could detect ambiguities and inconsistencies

Types of Semantics

Operational Semantics – the execution of the language is described directly.

Denotational Semantics – each phrase in the language is interpreted as a conceptual meaning that can be thought of abstractly.

Axiomatic Semantics – meaning to phrases is given by describing the logical axioms that apply to them.

Operational Semantics

Definition

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.

Operational Semantics

Definition

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.

To use operational semantics for a high-level language, a virtual machine is needed.

A hardware pure interpreter would be too expensive

A software pure interpreter also has problems:

- The detailed characteristics of the particular computer would make actions difficult to understand
- Such a semantic definition would be machine- dependent

Operational Semantics

A better alternative: A complete computer simulation

The process:

- Build a translator (translates source code to the machine code of an idealized computer)
- Build a simulator for the idealized computer

Evaluation of operational semantics:

- Good if used informally (language manuals, etc.)
- Extremely complex if used formally (e.g., VDL), it was used for describing semantics of PL/I.

Operational Semantics

Uses of operational semantics:

- Language manuals and textbooks
- Teaching programming languages

Two different levels of uses of operational semantics:

- Natural operational semantics
- Structural operational semantics

Evaluation

- Good if used informally (language manuals, etc.)
- Extremely complex if used formally (e.g.,VDL)

Denotational Semantics

Definition

Denotational Semantics is an approach of formalizing the meanings of programming languages by constructing mathematical objects (called denotations) that describe the meanings of expressions from the languages.

Denotational Semantics

Definition

Denotational Semantics is an approach of formalizing the meanings of programming languages by constructing mathematical objects (called denotations) that describe the meanings of expressions from the languages.

Based on recursive function theory

The most abstract semantics description method

Originally developed by Scott and Strachey (1970)

Denotational Semantics

The process of building a denotational specification for a language:

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

The meaning of language constructs are defined by only the values of the program's variables.

Denotational Semantics: program state

The state of a program is the values of all its current variables

$$s = <i_1, v_1>, <i_2, v_2>, \ldots, <i_n, v_n>$$

Let `VARMAP` be a function that, when given a variable name and a state, returns the current value of the variable

$$\mathtt{VARMAP}(i_j, s) = v_j$$

Decimal Numbers

```
<dec_num> → '0' | '1' | '2' | '3' | '4' | '5' |
          '6' | '7' | '8' | '9' |
            <dec_num> ('0' | '1' | '2' | '3' |
                       '4' | '5' | '6' | '7' |
                       '8' | '9')

Mdec('0') = 0,  Mdec ('1') = 1, ...,  Mdec ('9') = 9
Mdec (<dec_num> '0') = 10 * Mdec (<dec_num>)
Mdec (<dec_num> '1') = 10 * Mdec (<dec_num>) + 1
...
Mdec (<dec_num> '9') = 10 * Mdec (<dec_num>) + 9
```

Expressions

Map expressions onto $Z \cup \{\text{error}\}$

We assume expressions are decimal numbers, variables, or binary expressions having one arithmetic operator and two operands, each of which can be an expression

Expressions

```
Me(<expr>, s) δ=
    case <expr> of
      <dec_num> => Mdec(<dec_num>, s)
      <var> =>
           if VARMAP(<var>, s) == undef
                then error
                else VARMAP(<var>, s)
     <binary_expr> =>
          if (Me(<binary_expr>.<left_expr>, s) == undef
               OR Me(<binary_expr>.<right_expr>, s) =
                              undef)
               then error
   else
   if (<binary_expr>.<operator> == '+' then
      Me(<binary_expr>.<left_expr>, s) +
             Me(<binary_expr>.<right_expr>, s)
   else Me(<binary_expr>.<left_expr>, s) *
       Me(<binary_expr>.<right_expr>, s)
...
```

Assignment Statements

Maps state sets to state sets $\cup \{error\}$

```
Ma(x := E, s) $\delta$ =
    if Me(E, s) == error
       then error
       else s' =
       {<i1,v1'>,<i2,v2'>, ... ,<in,vn'>},
                where for j = 1, 2, ..., n,
                     if ij == x
                       then vj' = Me(E, s)
                       else vj' = VARMAP(ij, s)
```

Logical Pretest Loops

Maps state sets to state sets $\cup$ $\{error\}$

```
MI(while B do L, s) $\delta$ =
    if Mb(B, s) == undef
        then error
        else if Mb(B, s) == false
            then s
            else if MsI(L, s) == error
                then error
                else MI(while B do L, MsI(L, s))
```

Loop Meaning

The meaning of the loop is the value of the program variables after the statements in the loop have been executed the prescribed number of times, assuming there have been no errors

In essence, the loop has been converted from iteration to recursion, where the recursive control is mathematically defined by other recursive state mapping functions

- Recursion, when compared to iteration, is easier to describe with mathematical rigor

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

Because of its complexity, it are of little use to language users

Axiomatic Semantics

Based on formal logic (predicate calculus)

Original purpose: formal program verification

Axioms or inference rules are defined for each statement type in the language (to allow transformations of logic expressions into more formal logic expressions)

Definition

The logic expressions are called ***assertions***.

Axiomatic Semantics

Definition

An assertion before a statement (a ***precondition***) states the relationships and constraints among variables that are true at that point in execution.

Definition

An assertion following a statement is a ***postcondition***.

Definition

A ***weakest precondition*** is the least restrictive precondition that will guarantee the postcondition.

Axiomatic Semantics Form

Pre-, post form: `{P} statement {Q}`

An example:

```
a = b + 1 {a > 1}
```

One possible precondition: `{b > 10}`

Weakest precondition: `{b > 0}`

Program Proof Process

The postcondition for the entire program is the desired result:

- Work back through the program to the first statement. If the precondition on the first statement is the same as the program specification, the program is correct.

Axiomatic Semantics: Assignment

An axiom for assignment statements:

$$\texttt{(x = E): \{Q}_{x->E}\texttt{\} x = E \{Q\}}$$

The Rule of Consequence:

$$\frac{\texttt{\{P\} S \{Q\}, P' => P, Q => Q'}}{\texttt{\{P'\} S \{Q'\}}}$$

Axiomatic Semantics: Sequences

An inference rule for sequences of the form `S1; S2`

```
{P1} S1 {P2}
{P2} S2 {P3}
```

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

Axiomatic Semantics: Selection

An inference rules for selection

```
if B then S1 else S2
```

$$\frac{\texttt{\{B and P\} S1 \{Q\}, \{(not B) and P\} S2 \{Q\}}}{\texttt{\{P\} if B then S1 else S2 \{Q\}}}$$

Axiomatic Semantics: Loops

An inference rule for logical pretest loops:

```
{P} while B do S end {Q}
```

```
          (I and B) S {I}
-----------------------------------
{I} while B do S {I} and (not B)}
```

where `I` is the loop invariant (the inductive hypothesis)

Axiomatic Semantics: Axioms

Characteristics of the loop invariant: `I` must meet the following conditions:

`P => I`	the loop invariant must be true initially
`{I} B {I}`	evaluation of the Boolean must not change the validity of `I`
`{I and B} S {I}`	`I` is not changed by executing the body of the loop
`(I and (not B)) => Q`	if `I` is true and `B` is false, `Q` is implied
The loop terminates	can be difficult to prove

Loop Invariant

The loop invariant `I` is a weakened version of the loop postcondition, and it is also a precondition.

`I` must be weak enough to be satisfied prior to the beginning of the loop, but when combined with the loop exit condition, it must be strong enough to force the truth of the postcondition

Evaluation of Axiomatic Semantics

Developing axioms or inference rules for all of the statements in a language is difficult

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

Its usefulness in describing the meaning of a programming language is limited for language users or compiler writers

Denotation Semantics vs Operational Semantics

In operational semantics, the state changes are defined by coded algorithms

In denotational semantics, the state changes are defined by rigorous mathematical functions