DESCRIBING SEMANTICS

Semantics refers to the meaning associated with the statement in a programming language. It is all about the meaning of the statement which interprets the program easily. 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:

- ➢ Operational Semantics
- ➢ Denotational Semantics
- ➢ Axiomatic Semantics

1. 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

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.

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)

2. <u>Denotational Semantics</u>

Based on recursive function theory It is the The most abstract semantics description method, It was Originally developed by Scott and Strachey (1970)

<u>The process of building a denotational specification for a language:</u>

- ➢ 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

<u>Denotational Semantics: program state</u>

The state of a program is the values of all its current variables s = {<i1, v1>, <i2, v2>, …, <in, vn>}

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

VARMAP(ij, s) = vj

<u>Evaluation of Denotational Semantics</u>

- It can be used to prove the correctness of programs
- It provides a rigorous way to think about programs
- It can be an aid to language design
- It has been used in compiler generation systems
- Because of its complexity, it is of little use to language users

3. <u>Axiomatic Semantics</u>

- It is 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)
- The logic expressions are called assertions
- An assertion before a statement (a precondition) states the relationships and constraints among variables that are true at that point in execution
- An assertion following a statement is a postcondition
- A weakest precondition is the least restrictive precondition that will guarantee the postcondition

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

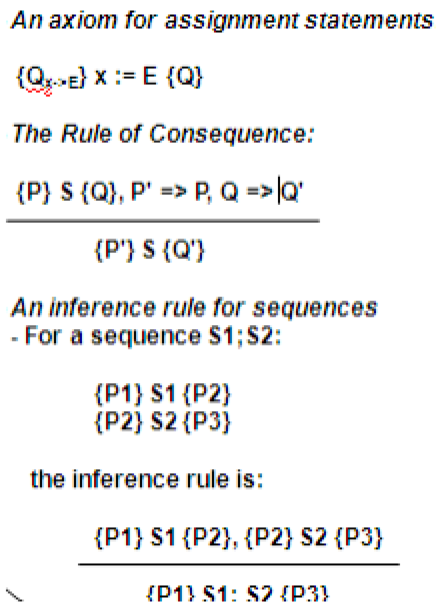
An example: a := b + 1 {a > 1}

One possible precondition: {b > 10}

Weakest precondition: {b > 0}

Program proof process: The post-condition for the whole program is the desired results. Work back through the program to the first statement. If the precondition on the first statement is the same as the program spec, the program is correct.

An axiom for assignment statements:



An axiom for assignment statements

$\{Q_{x\to E}\}$ x := E {Q}

The Rule of Consequence:

$$\frac{\{P\}\ S\ \{Q\},\ P' \Rightarrow P,\ Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

An inference rule for sequences
- For a sequence S1;S2:

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

the inference rule is:

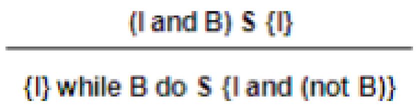
$$\frac{\{P1\}\ S1\ \{P2\},\ \{P2\}\ S2\ \{P3\}}{\{P1\}\ S1;\ S2\ \{P3\}}$$

An inference rule for logical pretest loops

For the loop construct:

{P} while B do S end {Q}

the inference rule is:



$$\frac{(I\ \text{and}\ B)\ S\ \{I\}}{\{I\}\ \text{while}\ B\ \text{do}\ S\ \{I\ \text{and}\ (\text{not}\ B)\}}$$

Where I is the loop invariant. Characteristics of the loop invariant

I must meet the following conditions:

1. P => I (the loop invariant must be true initially)

2. {I} B {I} (evaluation of the Boolean must not change the validity of I)

3. {I and B} S {I} (I is not changed by executing the body of the loop)

4. (I and (not B)) => Q (if I is true and B is false, Q is implied)

5. The loop terminates (this can be difficult to prove)

- 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