Semantic Description of Programming languages

Static versus Dynamic Semantics

- **Static Semantics** represents legal forms of programs that cannot be easily described using a BNF grammar. We call this semantics **static** because it is verified at compile time.
- **Dynamic Semantics** describes the meaning of programs or the effect caused by a program’s execution.

Why do we need to describe Dynamic Semantic?

- Programmers need to know exactly what each portion of their program does.
- People who design compilers must also know what each instruction must do.
- Even though they are not very precise, programmers and compiler designers must use descriptions in English given that formal semantic descriptions are very complex.
- Nonetheless, the definition of a formal and adequate notation is important since it can help compiler designers have access to more precise definitions, which could perhaps one day be the basis for the automatic generation of compilers.

How is (Dynamic) Semantics Described?

There exist three methods for describing Semantics:

- **Operational** Description: The meaning of a program is determined by the **execution** of its listing on a **virtual machine**.
- **Denotational** Description: The meaning of a program is described using functions that show the **effect** of applying a listing on the **state of the machine**.
- **Axiomatic** Description: the meaning of a program is described using **assertions** which specify the **constraints and relations** imposed by a listing.

Operational Semantics

- The idea behind operational semantics is to describe the meaning of a program by executing its instructions on a real or simulated machine. The changes that take place in the status of the machine when it executes these instructions represent the meaning of this instruction.
- In order to build an idealised simulated machine, two components are needed: a **translator** which translates low-level language, L, and a **virtual machine** whose status changes as the code written in language L is executed.
- Operational Semantics is effective. However, it is not formal and may cause circularities.

Denotational Semantics

- Denotational Semantics is the most rigourous description method for the semantic description of programs.
- The idea consists of defining, for each language entity, a **mathematical object** and **function** that attaches instances of this entity to instances of the corresponding mathematical object.
- Just like for operational semantics, the status of an idealised machine (in fact, the value of the variables) represents the meaning of an instruction.
- The difficulty of this method is in the creation of objects and functions for these objects. The notation is also difficult to read, albeit very concise.

Axiomatic Semantic I

- Axiomatic Semantic is defined along with a method of proof of program correctness.
- When the program is correct, there exists a proof of correctness and in this proof, each proposition is preceded and followed by a logical expression (***pre-condition*** and ***post-condition***) which specify the constraints on the program’s variables. It is these constraints that define the meaning of the program.

Axiomatic Semantic II

- The ***weakest pre-condition*** represents the least restrictive pre-condition which guarantees the correctness of the post-condition associated to the instruction of the program.
- If the weakest pre-condition can be calculated from the post-condition defined for each language instruction, proofs of correctness can be constructed for the programs of this language.
- The proofs are constructed starting from the end of a program and moving toward its begining.
- Axiomatic Semantic is not that useful with respect to the description of the meaning of programming languages because of its complexity. However, it is useful for research and for reasoning about programs.

Sémantique axiomatique III

- More specifically, the verification of a program's correctness is done in two steps:
  - The **association** of a formula with each step of the significant calculation.
  - The **demonstration** that the final formula **logically follows** from the initial formula thanks to the intermediate steps and formulae.
- The formulae for the assignment and the conditional statement are the basic formulae. The effect of all the other instructions is logically derived from the basic formulae.

Axiomatic Semantic IV: The Assignment Statement

- Let’s assume that ***x* = *E*** is an assignment statement and that ***Q*** is its post-condition. Then, its pre-condition is defined by the following axiom $P = Q_{x \dashrightarrow E}$ which means that P is computed just like Q with all the instances of x replaced by E.
- How can we prove the correctness of our programs (and in particular of an assignment statement) with such tools?

Axiomatic Semantic V: Justifying the procedure

- An assignment statement accompanied by its pre-condition and its post-condition can be considered to be a **theorem**.
- If the assignment statement’s axiom, when applied to the post-condition and to the assignment statement, produces the given pre-condition, then we can say that the **theorem has been proven**, and that, therefore, the program is correct.

Axiomatic Semantic VI: Consequence Rule (Narrowing and Widening)

- Sometimes, the precondition one obtains by the procedure just discussed does not correspond to the expected pre-condition.
- In such a case, one can use the consequence rule, which is given below:

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

Axiomatic SemanticVII: Sequences of instructions

Given two adjacent instructions with the following pre- and post- conditions:

{P1} S1 {P2}

{P2} S2 {P3}

The inference rule for such a sequence is:

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

Axiomatic SemanticsVIII: The Selection Statement

- ***If-then-else***:

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

- ***If-then:***

$$\frac{\{B \text{ and } P\}\ S1\ \{Q\},\ \{(\text{not } B) \text{ and } P\} \Rightarrow Q}{\{P\}\ \text{if } B \text{ then } S1\ \{Q\}}$$

Axiomatic Semantics IX: Loops with pre-test

- In a loop with a pre-test (also known as a while loop), there is a repetition of instructions. The problem with these loops, however, is that it is impossible to know, ahead of time, how many repetitions there will be. Therefore, it is quite difficult to determine exactly the correctness of these loops.
- The method we use is similar to the ***<u>mathematical method of induction</u>***.
- The inductive hypothesis is called the ***<u>loop invariant</u>***.

Axiomatic Semantics X: Loops with Pre-test

- The inference rule that allows us to find the pre-consition of a while loop is the following:

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

I represents the loop invariant but it is not provided. We must find it!

How? By calculating the pre-condition for a certain number of repetitions and trying to guess a pattern.

Axiomatic Semantics XI: Loops with Pre-test

- Finding the loop invariant is not all!!!!!
- Given instruction ***{P} while B do S end {Q}***, and loop invariant, ***I*** , here is a summary of all that needs to be proven in order to prove the correctness of a while loop:

P => I

{I} B {I}

{I and B} S {I}

(I and (not B)) => Q

The loop terminates