Journal of Advanced Life Sciences and Technology
ISSN 3095-1724
Volume 12 Issue 4
October– December 2024
Impact Factor: 7.06
Enquiry: contact@continentalpub.online
Published by Continental Publication | https://continentalpub.online/index.php/life-sciences

INVESTIGATING OPERATIONAL SEMANTICS: THE FUNDAMENTALS OF PROGRAMMING

Hiroshi Yamamoto
School of Computing, University of Tokyo, Tokyo, Japan

Abstract: *Computer programming is a fundamental tool in the world of computerization, and a solid grasp of its foundational aspects, including syntax, semantics, and their implications, is essential. Syntax addresses the grammatical structure of a program, while semantics delves into the meaning of programs that adhere to grammatical correctness. Consider, for instance, the statement: "c: =a; a:=b; b:=c" (Expression(i)). A syntactic analysis of Expression (i) reveals three distinct statements, each separated by semicolons. Each statement consists of a variable, an assignment operator (":="), and an expression, which is also a variable. In contrast, the semantics of this statement dictates the exchange of values between variables a and b, with c ultimately receiving the value of b (Nielson and Nielson, 2007). Semantics, in the context of programming languages, encompasses the computational meaning of each program. This field is deeply involved in the rigorous mathematical examination of programming language meanings and models of computation (Moses, 2006). Semantics serves multiple purposes, from understanding the intricacies of specific programming languages to establishing a foundation for verifying the properties of particular programs. Additionally, it facilitates the expression of design choices and provides insights into the interaction between various language features (Sewell, 2008). This paper explores the vital concepts of syntax and semantics in computer programming, shedding light on their significance and practical applications.*

Keywords: *Computer programming, syntax, semantics, program analysis, programming language semantics.*

1.1 Introduction

Programming is a major tool for computerization and there is need to understand its basics; the how, the effects and the assertions. The syntax of a program deals with the grammatical structure of the program while the semantics deals with the meaning of grammatically correct programs. For instance, consider the following statement: c:=a; a:=b; b:=c Expression(i)

A syntactic analysis of the program statement given in Expression (i) above consists of three statements separated by ";". Each of these statements has a variable, followed by an assignment statement,":=", and an expression which is also a variable. Whereas, the semantics of the statement expresses that the program is to exchange the values of variables a and b and setting c to the final value of b (Nielson and Nielson, 2007).

A semantics for a programming language models the computational meaning of each program (Moses, 2006). It is also concerned with the rigorous mathematical study of the meaning of programming languages and models of computation. Semantics can be used to understand a particular language and as a foundation for

Vol. 12 No. 4 | Imp. Factor: 7.061
DOI: https://doi.org/10.5281/zenodo.14604950

proving properties of particular programs. It can also be used as a tool for expressing design choices, understanding language features and how they interact (Sewell, 2008).

There are three major levels of semantics namely static semantics which models compile-time checks, dynamic semantics which models run-time behaviour and semantic equivalences between programs which may abstract from details of models (Moses, 2006). Dynamic semantics is further subdivided into operational semantics, denotational semantics and axiomatic semantics (Nielson and Nielson, 2007; Hennessy, 1991). While operational semantics deals with how the effect of a computation is produced, denotational semantics models the meanings by mathematical objects that represent the effect of executing the constructs and axiomatic semantics deals with the specific properties of the effect of executing the constructs. The formal semantics of a language is given by a mathematical model that describes the possible computations described by the language. It is concerned with rigorously specifying the meaning, or behaviour, of programs and pieces of hardware among others (Nielson and Nielson, 2007; Plotkin, 1982).

Formal semantics is capable of revealing ambiguities and also forms the basis for implementation, analyses and verification of programs.

2.1 Basic Concepts of Operational Semantics (OS)

An operational explanation of the meaning of a construct tells how to execute it (Abramsky and Hankin, 1987; Aho, Sethi and Ullman, 1986; Jones, 1980). To execute a sequence of statements separated by „;“ as seen in Expression (i), the individual statements are being executed one after the other and from left to right. To execute a statement consisting of a variable followed by the assignment operator „:=„ and another variable, the value of the second variable is determined and assigned to the first variable. The execution of a program in a state where **a** has the value **3**, **b** the value **5** and **c** the value **0** is done by the following derivation sequence:

(i). c:=a; a:=b; b:=c, [a→3, b→5, c→0] 〉 (ii). a:=b; b:=c, [a→3, b→5, c→3] 〉 (iii). b:=c, [a→5, b→5, c→3] 〉
(iv). [a→5, b→3, c→3] 〉

In the first step, the statement c:=a is executed and the value of c is changed to 3 whereas those of a and b are unchanged. The remaining program is now a:=b; b:=c. After the second step, the value of a is 5 and we are left with the program b:=c. The third and final step of the computation changes the value of b to 3. Hence, the initial values of a and b have been exchanged, using c as a temporary variable. When this kind of operational semantics is formalized, it is often referred to as structural operational semantics (or small-step semantics). An alternative operational semantics is called natural semantics (or big-step semantics) and it differs from the structural operational semantics by hiding more execution details. Figures 1 and 2 show the rules for both structural operational semantics and natural semantics respectively.

2.2 Structural Operational Semantics (SOS)

Structural operational semantics (SOS) provides a framework to give an operational semantics to programming and specification languages. SOS generates a labelled transition system, whose states are the closed terms over an algebraic signature, and whose transitions between states are obtained inductively from a collection of so-called transition rules of the form: $\frac{\text{premises}}{\text{conclusion}}$

Structural operational semantics provides transition rules for the evaluation of expressions and execution of commands as seen in Figure 1. If the number of premises is zero, then, the line is omitted, and we refer to the rule as an axiom. (Aceto, Fokkinky and Verhoefz, 2001; Slonneger and Kurtz, 1995).

Journal of Advanced Life Sciences and Technology
Vol. 12 No. 4 | Imp. Factor: 7.061
DOI: https://doi.org/10.5281/zenodo.14604950

(1) [ass] ⟨x:=a,s⟩ → s(x→A[a]s) sos
(2) [skip] ⟨skip, s⟩ → s
sos

$$\frac{\langle S , s\rangle \rightarrow \langle S' , s'\rangle}{}$$

1 1
(3) [comp ¹] ⟨S ; S , s⟩ → ⟨S' ; S , s'⟩ sos 1 2 1 2

$$\frac{\langle S , s\rangle \rightarrow s'}{}$$

1
(4) [comp²] ⟨S ; S , s⟩ → ⟨ S , s'⟩ sos 1 2 2

tt
(5) [if] ⟨if b then S else S ,s⟩ → ⟨S , s⟩ = if B[b]s = tt sos 1
2 1 ff
(6) [if] ⟨if b then S else S ,s⟩ → ⟨S , s⟩ = if B[b]s = ff sos 1
2 2
(7) [while] ⟨while b do S, s⟩ → ⟨if b then (S; while b do S) else skip s ⟩
sos
Note: sos means Structural Operational Semantics.
Figure 1: Structural Operational Semantics (Source: Nielson and Nielson, 1999).

The role of a statement in „While" is to change the state. Given that x is bound to 5 in a state, s, and the statement x := x + 1 is executed, then, a new state, s_0, where x is bound to 6 is produced. So, while the semantics of arithmetic and boolean expressions only inspect the state in order to determine the value of the expression, the semantics of statements will modify the state as well. (Nielson and Nielson, 1999; Despeyroux, 1986).

For the language „While", one can easily specify both kinds of operational semantics and they will still be equivalent. For the two kinds, the meaning of statements is specified by a transition system which has two types of configurations as shown below:

⟨S, s⟩ representing that the statement S is to be executed from the state s, and s representing a final state.

The transition relation will then describe how the execution takes place. The difference between the two approaches to operational semantics amounts to different ways of specifying the transition relation. An example of how SOS specifies the translation relation is given below:

Consider the same example earlier given in Expression (i): (c:=a; a:=b); b:=c

Let s_0 be the state that maps all variables except a and b to **0**

Let s_0 a =**3** and s_0 b =**5**

Then, the derivation sequence is as follows:

(i) ⟨(c:=a; a:=b); b:=c, s_0⟩
(ii) ⟨a:=b; b:=c, s_0[c→3]⟩
(iii) ⟨b:=c, (s_0[c→3])[a→5]⟩ (iv) ⟨((s_0[c→3])[a→5])[b→3]⟩

Each of the above steps has corresponding trees that explain why they take place. For step (i):

⟨(c:=a; a:=b); b:=c, s_0⟩ → ⟨a:=b; b:=c, s_0[c→3]⟩

DOI: https://doi.org/10.5281/zenodo.14604950

The derivation tree is shown below:

$$\frac{\langle c:=a, s_0\rangle \rightarrow s_0[c \rightarrow 3]}{\langle c:=a; a:=b), s_0\rangle \rightarrow \langle a:=b, s_0[c \rightarrow 3]\rangle}$$

$$\langle (c:=a; a:=b); b:=c, s_0\rangle \rightarrow \langle a:=b; b:=c, s_0[c \rightarrow 3]\rangle$$

The above tree has been constructed from the axiom [ass] and [comp^1] (from Figure 1). Hence, it is seen here that

sos sos details of execution are explained and this is why SOS is called small-step semantics.

2.3 Natural Semantics (NS)

In a natural semantics, the relationship between the initial and the final state of an execution is of utmost concern. As mentioned earlier, it is defined as a binary relation between configurations as explained under SOS. Transitions from the initial pair to the terminal state are denoted by:

$\langle S, s\rangle \rightarrow s'$.

The execution of S from s will terminate and the resulting state will be s'. Natural Semantics is defined by the set of derivations or rules shown in Figure 2 (Nielson and Nielson, 1999; Bakel, 2002). To show the translation relation of natural semantics, consider the same example earlier given in Expression (i): (c:=a; a:=b); b:=c Let s_0 be the state that maps all variables except a and b to **0**

Let s_0 a =**3** and s_0 b =**5**

Then, the derivation sequence is as follows:

$$\frac{\langle c:=a, s_0\rangle \rightarrow s_1 \qquad \langle a:=b, s_1\rangle \rightarrow s_2}{\langle c:=a; a:=b), s_0\rangle \rightarrow s_2 \qquad \langle b:=c, s_2\rangle \rightarrow s_3}$$

$$\langle (c:=a; a:=b); b:=c, s_0\rangle \rightarrow s_3$$

From the above derivation, the following abbreviations are used: $s_1 = s_0[c \rightarrow 3]$ $s_2 = s_1[a \rightarrow 5]$ $s_3 = s_2[b \rightarrow 3]$

The derivation tree has three leaves denoted as: $\langle c:=a, s_0\rangle \rightarrow s_1$, $\langle a:=b, s_1\rangle \rightarrow s_2$ and $\langle b:=c, s_2\rangle \rightarrow s_3$, corresponding to the tree applications of the axiom [ass_{ns}].

DOI: https://doi.org/10.5281/zenodo.14604950

(1) [ass]ns $\langle x:=a,s\rangle \rightarrow s(x \rightarrow A[a]s)$

(2) [skip]ns $\langle skip, s\rangle \rightarrow s$

(3) [comp]ns $\dfrac{\langle S , s\rangle \rightarrow s', \ \langle S , s'\rangle \rightarrow s''}{\langle S ; S , s\rangle \rightarrow s''}$ (1 2)

(4) tt [if]ns $\dfrac{\langle S , s\rangle \rightarrow s'}{\langle \text{if } b \text{ then } S \text{ else } S , s\rangle \rightarrow s'}$ (1 2) if B[b]s = tt

(5) ff [if]ns $\dfrac{\langle S , s\rangle \rightarrow s'}{\langle \text{if } b \text{ then } S \text{ else } S , s\rangle \rightarrow s'}$ (1 2) if B[b]s = ff

(6) tt [while]ns ff $\dfrac{\langle S , s\rangle \rightarrow s', \ \langle \text{while } b \text{ do } S, s'\rangle \rightarrow s''}{\langle \text{while } b \text{ do } S, s\rangle \rightarrow s''}$ (1) if B[b]s = tt

(7) [while]ns $\langle \text{while } b \text{ do } S, s\rangle \rightarrow s$ if B[b]s = ff

Note: ns means Natural Semantics.

Figure 2: Natural Semantics (Source: Nielson and Nielson, 1999).

The rule [comp$_{ns}$] has been applied twice. One instance is:

~~$\langle c:=a, s_0\rangle \rightarrow s_1, \ \langle a:=b, s_1\rangle \rightarrow s_2$~~

$\langle c:=a; a:=b), s_0\rangle \rightarrow s_2$

This instance has been used to combine the leaves $\langle c:=a, s_0\rangle \rightarrow s_1$ and $\langle a:=b, s_1\rangle \rightarrow s_2$ with the internal node labelled $\langle c:=a; a:=b), s_0\rangle \rightarrow s_2$. The other instance is:

$$\frac{\langle c:=a; a:=b), s_0\rangle \rightarrow s_2, \ \langle b:=c, s_2\rangle \rightarrow s_3}{\langle (c:=a; a:=b); b:=c, s_0\rangle \rightarrow s_3}$$

This instance has been used to combine the internal node $\langle c:=a; a:=b), s_0\rangle \rightarrow s_2$ and the leaf $\langle b:=c, s_2\rangle \rightarrow s_3$ with the root $\langle (c:=a; a:=b); b:=c, s_0\rangle \rightarrow s_3$.

Hence, this example shows that, unlike SOS, NS actually hides certain details and thus, the name big-step semantics. The transition between states is of utmost concern here. Despite the difference in the specification of the transition relation used, both SOS and NS gave equivalent results. Also, the

Vol. 12 No. 4 | Imp. Factor: 7.061
DOI: https://doi.org/10.5281/zenodo.14604950

examples given also affirms that Formal semantics helps to proof the correctness of programs. (Ganor and Juhasz , 2007).

3. Conclusion

This paper has been able to show the interest of operational semantics by enumerating how the effect of a computation is produced. While structural operational semantics has described how the individual steps of the computations take place, natural semantics has described how the overall results of execution are obtained. I hereby recommend that "Formal Semantics" should be taken as a course by computer science students in tertiary institutions so as to enhance a better performance in their programming work thereby enhancing the production of indigenous software that meets the specific needs of the people in our community.

References

Abramsky, S. and Hankin, C. (1987).Abstract Interpretation of Declarative Languages, Ellis Horwood.

Aceto, L., Fokkinky, W. and Verhoefz, C. (2001). Structural Operational Semantics

Aho, A.V., Sethi, R. and Ullman, J.D. (1986). Compilers: Principles, Techniques and Tools, Addison-Westley.

Bakel, S. V. (2002). Operational Semantics.Course Notes. Department of Computing Imperial College of Science, Technology and Medicine

Despeyroux, J. (1986). Proof of translation in natural semantics, *Proceedings ofsymposium on logic in Computer Science,* Cambridge, Massachusetts, USA.

Ganor , R. and Juhasz , U. (2007). Operational Semantics.Class notes for a lecture given by MoolySagiv, Tel Aviv University, 24/5/2007

Hennessy, M. (1991). The Semantics of Programming Languages: An Elementary Introduction using Structural Operational Semantics, Wiley.

Jones, C.B. (1980). Software Development: A Rigorous Approach, Prentice-Hall.

Moses, P. D. (2006). Formal Semantics of Programming language.Electronic Books in Theoretical Computer Science. 148 (2001) 41-73.

Nielson, H.R. and Nielson F. (1999). Semantics With Applications: A Formal Introduction.

Nielson, H. R.and Nielson, F. (2007). *Semantics with Applications: An Appetizer*.

Plotkin, G.D. (1982). An Operational Semantics for CSP, in: Formal Description of Programming Concepts II, *Proceedings of TC-2 Work. Conf. (ed. Bjorner D.),* North Holland.

Sewell, P. (2008). Semantics of Programming Languages.Computer Science Tripos, Part 1B.2008–9, Computer Laboratory, University of Cambridge.

Slonneger, K. and Kurtz, B. L. (1995). Formal Syntax and Semantics of Programming Languages: A Laboratory-Based Approach. Addison-Wesley, Reading, Massachusetts.

DOI: https://doi.org/10.5281/zenodo.14604950

Springer. ISBN 978-1-84628-692-6.