Semantics with Applications
Introduction

Pablo López

University of Málaga

September 10, 2024

Outline

Introduction

Semantic Description Methods

Outline

Introduction

Semantic Description Methods

What is Formal Semantics

Formal Semantics is concerned with **rigorously** specifying the **meaning**, or behavior, of programs, pieces of hardware, etc.

The Need for Formal Semantics

- can reveal ambiguities and subtle complexities
- basis for practical tools (implementations, analyzers, verifiers, etc.)

Outline

Introduction

Semantic Description Methods

Syntax vs Semantics (I)

Syntax is concerned with the **grammatical** structure of programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the syntax of a language with only one statement: assignment of a variable to a variable.

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.
What did you get?

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.
What did you get?
What about other valid program?

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.

What did you get?

What about other valid program?

Two powerful ideas:

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.

What did you get?

What about other valid program?

Two powerful ideas:

Semantics is **syntax-directed**: gives meaning to every syntactic construct (;, :=, etc.).

Syntax vs Semantics (II)

Semantics is concerned with the **meaning** of grammatically correct programs

```
    z:= x;
    x:= y;
    y:= z
```

Exercise. Describe the meaning of the previous program.

What did you get?

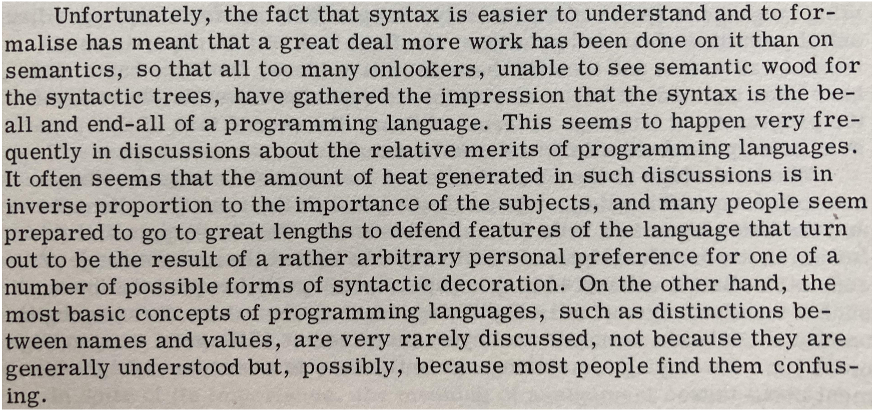
What about other valid program?

Two powerful ideas:

Semantics is **syntax-directed**: gives meaning to every syntactic construct (`;`, `:=`, etc.).

Semantics is **compositional**: the meaning of the program can be obtained from the meaning of its constructs.

Strachey on Syntax vs Semantics



Unfortunately, the fact that syntax is easier to understand and to formalise has meant that a great deal more work has been done on it than on semantics, so that all too many onlookers, unable to see semantic wood for the syntactic trees, have gathered the impression that the syntax is the be-all and end-all of a programming language. This seems to happen very frequently in discussions about the relative merits of programming languages. It often seems that the amount of heat generated in such discussions is in inverse proportion to the importance of the subjects, and many people seem prepared to go to great lengths to defend features of the language that turn out to be the result of a rather arbitrary personal preference for one of a number of possible forms of syntactic decoration. On the other hand, the most basic concepts of programming languages, such as distinctions between names and values, are very rarely discussed, not because they are generally understood but, possibly, because most people find them confusing.

Why Formal Semantics?

- most languages are described by informal semantics (prose and examples)
- **example**: the `match` statement in Python

Why Formal Semantics?

- most languages are described by informal semantics (prose and examples)
- **example**: the `match` statement in Python

What's wrong with it?

Semantic Styles

Three major approaches to semantics:

- Operational Semantics (Khan, Plotkin)
- Denotational Semantics (Strachey, Scott)
- Axiomatic Semantics (Floyd, Hoare, Dijkstra)

Other approaches (game semantics, evolving algebras, etc.) are available.

Goal of the Course

Illustrate:

- fundamental ideas
- applications
- implementation

of the operational semantics approach using the simple imperative language `WHILE`.

First, let's briefly review the operational, denotational, and axiomatic semantics.

Two Styles of Operational Semantics

- Structural Operational Semantics (*small step*, Gordon Plotkin)

- Natural Semantics (*big step*, Gilles Kahn)

Operational Semantics

- The meaning of a construct is specified by the computation it induces when it is executed on a (abstract) machine.
- In particular, it is of interest **how** the effect of the computation is produced.
- The operational semantics is rather **independent** of machine architectures and implementation strategies.

Operational Semantics: Example

For every construct we describe **how** it is executed.

```
    z:= x;
    x:= y;
    y:= z
```

- semicolon separated statements are executed sequentially, left to right
- assignments are executed replacing the value of the variable on the left by the value of the variable on the right

Structural Operational Semantics: Derivation Sequence

$$\begin{array}{lr}
 & \langle \mathrm{z{:=}x;\ x{:=}y;\ y{:=}z}, \quad [\mathrm{x}\mapsto\mathbf{5},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{0}]\rangle \\
\Rightarrow & \langle \mathrm{x{:=}y;\ y{:=}z}, \quad [\mathrm{x}\mapsto\mathbf{5},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{5}]\rangle \\
\Rightarrow & \langle \mathrm{y{:=}z}, \quad [\mathrm{x}\mapsto\mathbf{7},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{5}]\rangle \\
\Rightarrow & [\mathrm{x}\mapsto\mathbf{7},\ \mathrm{y}\mapsto\mathbf{5},\ \mathrm{z}\mapsto\mathbf{5}]
\end{array}$$

Natural Semantics: Derivation Tree

$$\dfrac{\dfrac{\langle \mathrm{z}{:=}\mathrm{x},\ s_0\rangle \to s_1 \qquad \langle \mathrm{x}{:=}\mathrm{y},\ s_1\rangle \to s_2}{\langle \mathrm{z}{:=}\mathrm{x};\ \mathrm{x}{:=}\mathrm{y},\ s_0\rangle \to s_2} \qquad \langle \mathrm{y}{:=}\mathrm{z},\ s_2\rangle \to s_3}{\langle \mathrm{z}{:=}\mathrm{x};\ \mathrm{x}{:=}\mathrm{y};\ \mathrm{y}{:=}\mathrm{z},\ s_0\rangle \to s_3}$$

where:

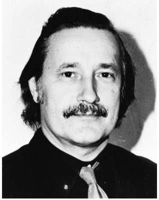
$$\begin{array}{lcl}
s_0 & = & [\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{0}] \\
s_1 & = & [\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{5}] \\
s_2 & = & [\mathrm{x}{\mapsto}\mathbf{7},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{5}] \\
s_3 & = & [\mathrm{x}{\mapsto}\mathbf{7},\ \mathrm{y}{\mapsto}\mathbf{5},\ \mathrm{z}{\mapsto}\mathbf{5}]
\end{array}$$

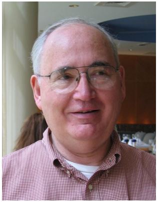
Applications of Operational Semantics

- **Natural semantics:** the SML Programming Language
- **Structural operational semantics:** the WebAssembly Programming Language

The Scala Programming Language is formally described by the DOT Calculus.

Denotational Semantics: the Strachey-Scott Approach





Denotational Semantics

- Meanings are modelled by mathematical objects that represent the **effect** of executing the constructs. Thus **only the effect** is of interest, not **how** it is obtained.
- Denotational semantics is concerned with **what** is computed, not **how**.
- By abstracting away from execution details it becomes **easier to reason** about programs: it simply amounts to reasoning about mathematical objects.
- Can deal with program properties other than execution behavior (basis for static analyzers).

Denotational Semantics

- Meanings are modelled by mathematical objects that represent the **effect** of executing the constructs. Thus **only the effect** is of interest, not **how** it is obtained.
- Denotational semantics is concerned with **what** is computed, not **how**.
- By abstracting away from execution details it becomes **easier to reason** about programs: it simply amounts to reasoning about mathematical objects.
- Can deal with program properties other than execution behavior (basis for static analyzers).

Is it that simple?

Denotational Semantics: Example

For every construct we define a **function** that computes its **effect** when executed.

```
    z:= x;
    x:= y;
    y:= z
```

- the effect of semicolon separated statements is the **functional composition** of the effects of the individual statements
- the effect of an assignment statement is a **function** that given a state returns a new state identical to the original, except that the value of the variable on the left is replaced by the value of the variable on the right

Denotational Semantics: Function Application

A function for each statement:

$$\mathcal{S}[\![\mathrm{z} := \mathrm{x}]\!] \qquad \mathcal{S}[\![\mathrm{x} := \mathrm{y}]\!] \qquad \mathcal{S}[\![\mathrm{y} := \mathrm{z}]\!]$$

A function for the overall program (beware the order):

$$\mathcal{S}[\![\mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}; \mathrm{y} := \mathrm{z}]\!] = \mathcal{S}[\![\mathrm{y} := \mathrm{z}]\!] \circ \mathcal{S}[\![\mathrm{x} := \mathrm{y}]\!] \circ \mathcal{S}[\![\mathrm{z} := \mathrm{x}]\!]$$

Applying the function to the initial state yields the effect:

$$\begin{aligned}
&\mathcal{S}[\![\mathrm{z}{:=}\mathrm{x};\ \mathrm{x}{:=}\mathrm{y};\ \mathrm{y}{:=}\mathrm{z}]\!]([\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{0}])\\
&\qquad = (\mathcal{S}[\![\mathrm{y}{:=}\mathrm{z}]\!] \circ \mathcal{S}[\![\mathrm{x}{:=}\mathrm{y}]\!] \circ \mathcal{S}[\![\mathrm{z}{:=}\mathrm{x}]\!])([\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{0}])\\
&\qquad = \mathcal{S}[\![\mathrm{y}{:=}\mathrm{z}]\!](\mathcal{S}[\![\mathrm{x}{:=}\mathrm{y}]\!](\mathcal{S}[\![\mathrm{z}{:=}\mathrm{x}]\!]([\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{0}])))\\
&\qquad = \mathcal{S}[\![\mathrm{y}{:=}\mathrm{z}]\!](\mathcal{S}[\![\mathrm{x}{:=}\mathrm{y}]\!]([\mathrm{x}{\mapsto}\mathbf{5},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{5}]))\\
&\qquad = \mathcal{S}[\![\mathrm{y}{:=}\mathrm{z}]\!]([\mathrm{x}{\mapsto}\mathbf{7},\ \mathrm{y}{\mapsto}\mathbf{7},\ \mathrm{z}{\mapsto}\mathbf{5}])\\
&\qquad = [\mathrm{x}{\mapsto}\mathbf{7},\ \mathrm{y}{\mapsto}\mathbf{5},\ \mathrm{z}{\mapsto}\mathbf{5}]
\end{aligned}$$

Denotational Semantics Applications

Foundation of many static analyzers:

- Determine whether variables have been initialized
- Replace constant expressions by their values
- Eliminate dead code

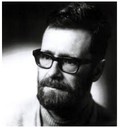
The main drawback is that it requires a firm mathematical basis which is far from trivial for some constructs.

Three Styles of Axiomatic Semantics

- Strongest Verifiable Consequent (Robert W. Floyd)

- Hoare Logic (C.A.R. Hoare)

- Weakest Precondition (Edsger W. Dijkstra)



Axiomatic Semantics

- Specific **properties** of the effect of executing the constructs are expressed as **assertions**; there might be aspects of the execution that are ignored.
- The axiomatic semantics provides an easy way of proving properties (partial correctness, total correctness, requirements, contracts, execution time) of programs.

Axiomatic Semantics: Example

For every construct we define a **logical rule** that reflects its effect on the properties (**preconditions** and **postconditions**) when executed.

```
    z:= x;
    x:= y;
    y:= z
```

- for semicolon separated statements, the postcondition of the preceding statement must be the precondition of the subsequent statement
- for assignment statements, the precondition is identical to the postcondition except that the variable on the left is replaced by the expression on the right

Axiomatic Semantics: Proof Tree

$$\frac{\dfrac{\{\ p_0\ \}\ \texttt{z:=x}\ \{\ p_1\ \} \qquad \{\ p_1\ \}\ \texttt{x:=y}\ \{\ p_2\ \}}{\{\ p_0\ \}\ \texttt{z:=x; x:=y}\ \{\ p_2\ \}} \qquad \{\ p_2\ \}\ \texttt{y:=z}\ \{\ p_3\ \}}{\{\ p_0\ \}\ \texttt{z:=x; x:=y; y:=z}\ \{\ p_3\ \}}$$

where:

$$\begin{aligned}
p_0 &= \texttt{x=n} \wedge \texttt{y=m} \\
p_1 &= \texttt{z=n} \wedge \texttt{y=m} \\
p_2 &= \texttt{z=n} \wedge \texttt{x=m} \\
p_3 &= \texttt{y=n} \wedge \texttt{x=m}
\end{aligned}$$

Axiomatic Semantics: Specification vs Implementation

```
{x = n && y = m}
   z:= x; x:= y; y:= z
{x = m && y = n}

{x = n && y = m}
   if (x=y) then
      skip
   else
      (z:= x; x:= y; y:= z)
{x = m && y = n}

{x = n && y = m}
   while true do
      skip
{x = m && y = n}
```

Applications of Axiomatic Semantics

- the Dafny Programming and Verification Language
- the Facebook Infer Tool
- the CompCert C Verified Compiler
- the seL4 Verified Microkernel

Quiz

Why so many different approaches to semantics? Is there one that rules them all?

Quiz

Why so many different approaches to semantics? Is there one that rules them all?

No; they serve different purposes:

- Operational: guides implementers of interpreters and compilers
- Denotational: guides implementers of analyzers (dead code, security holes,. . .)
- Axiomatic: guides programmers to nirvana (i.e. correct code)