CS4022D Principles of Programming Languages
Lecture #4: Semantics - Part 2

Saleena N
NIT Calicut

August 04, 2021

Semantics

- Meaning of syntactically valid programs
- Mostly described informally
- Approaches to Formal Semantics
  - Operational, Denotational, Axiomatic

Semantics: Boolean Expressions

Evaluation of *expr*1 && *expr*2

- Short-Circuit Evaluation Semantics
  - if *expr*1 evaluates to *false*, *expr*2 is not evaluated
- Complete Evaluation Semantics
  - both *expr*1 and *expr*2 are evaluated

Different Semantics: Different run-time behavior

Evaluation of $f(x)$ && $g(y)$

- Short-Circuit Evaluation Semantics
  - if $f(x)$ evaluates to *false*, $g(y)$ is not evaluated
- Complete Evaluation Semantics
  - suppose $g(y)$ results in a run-time error???

Formal Semantics: Approaches

- Operational semantics
- Denotational semantics
- Axiomatic semantics

Operational Semantics

- meaning of a construct is specified by the computation it induces when it is executed on a machine (mostly an abstract machine).
- the interest is on how the effect of a computation is produced.
- meaning of $x = y + z$?
  - evaluate the expression $y + z$ in the current state[1]
  - assign the value to variable x, resulting in a new state [2]
- meaning of $x = y + z; a = x$?

---

[1] Program state: Mapping of values to variables (simple view)

[2] Assignment causes a change in state (side effect)

Operational Semantics

- Meaning of expression (1+2)*(3+4)

  $\underline{(1+2)} * (3+4) \rightarrow 3 * \underline{(3+4)} \rightarrow \underline{3 * 7} \rightarrow 21$

  sequence of internal steps of computation
- *Intensional Semantics* - sequence of internal steps of computation is important

Operational Semantics

- two different Operational Semantics for (1+2)*(3+4)

$$\underline{(1+2)} * (3+4) \rightarrow 3 * \underline{(3+4)} \rightarrow \underline{3 * 7} \rightarrow 21$$

$$(1+2) * \underline{(3+4)} \rightarrow \underline{(1+2)} * 7 \rightarrow \underline{3 * 7} \rightarrow 21$$

- Factorial function - differently coded functions may have different semantics

Boolean Expression - Operational Semantics

- Language Syntax

$$
\begin{aligned}
B ::= &\ true \\
&\ false \\
&\ B \wedge\ B
\end{aligned}
$$

Boolean Expression - Operational Semantics

- Language Syntax

  $B ::= true$

  $\quad\quad false$

  $\quad\quad B \wedge B$

- Some sentences (strings) in the language:

  true

  false

  $true \wedge false$

  $true \wedge false \wedge false$

  $false \wedge false \wedge false \wedge true$

Boolean Expression - Operational Semantics

Evaluation of an expression of the form $B_1 \wedge B_2$

- if B_1 is *false*, the entire expression evaluates to *false*
- if B_1 is *true*, evaluate B_2
- if B_1 is not a value, evaluate B_1 to a value say v_1 and then evaluate $v_1 \wedge B_2$

Boolean Expression - Semantics

$$\mathit{false} \;\wedge\; B \rightarrow \mathit{false}$$

$$\mathit{true} \;\wedge\; B \rightarrow B$$

$$\frac{B_1 \rightarrow B_1'}{B_1 \;\wedge\; B_2 \rightarrow B_1' \;\wedge\; B_2}$$

Boolean Expression - Operational Semantics

$$true \ \wedge \ B \rightarrow B$$

$$false \ \wedge \ B \rightarrow false$$

$$\frac{B_1 \rightarrow B_1'}{B_1 \ \wedge \ B_2 \rightarrow B_1' \ \wedge \ B_2}$$

- Meaning of $(true \wedge true) \wedge false$?

$$(true \wedge true) \wedge false \quad \rightarrow \quad true \wedge false \quad \rightarrow \quad false$$

- as steps of evaluations in an abstract machine
- evaluating to final value *false*

Denotational Semantics

- Meaning is a mathematical function from input data to output data
- Expression Semantics
  $\mathcal{M}$: *Expression* $\rightarrow$ *Value*
  $\mathcal{M}(1+2)*(3+4) = 21$

Denotational Semantics

- Operational steps are not important
- *Extensional Semantics* - only the relationship between input and output is important
- Factorial function - differently coded functions have the same semantics

Booleans: Denotational Semantics

- Define a Semantic function for each syntactic category

$$
\begin{aligned}
\mathcal{B}[\![\mathit{true}]\!] &= \mathit{true} \\
\mathcal{B}[\![\mathit{false}]\!] &= \mathit{false} \\
\mathcal{B}[\![b_1 \wedge b_2]\!] &= \mathcal{B}[\![b_1]\!] \wedge \mathcal{B}[\![b_2]\!]
\end{aligned}
$$

Axiomatic Semantics

- History
  - R. W. Floyd *Assigning Meanings to Programs*[1967] - rules for reasoning about flow charts, fragments of ALGOL
  - C. A. R. Hoare *An axiomatic basis for Computer Programming*[1969] -rules for reasoning about programming languages, partial correctness of programs. Suggested axiomatic techniques for the definition of programming language semantics
- name axiomatic semantics?
  - meaning of a construct specified in terms of axioms saying how to prove properties of it.
- method for proving properties and for explaining the meaning of program constructs
- more abstract than denotational and operational definitions

Axiomatic Semantics

- Formal proof of program properties - proof of partial correctness
- Properties as *assertions* of the form $\{P\}\ S\ \{Q\}$ (P - precondition, Q - post condition)
- if P holds in the initial state, and if the execution of S terminates when started in that state, then Q will hold in the state in which S halts

$$\{a > 0\}\ a := a - 1\ \{a \geq 0\}$$

Partial correctness - Example

$\{x = n \wedge \ n > 0\}$

$y := 1; \ while \ (x! = 1) \ \{y := x * y; \ x := x - 1\}$

$\{y = n!\}$

Conclusion

- Formalizing Semantics
  - Language Designers, Compiler Writer, Programmer
  - Standard for implementer - Program should behave same in different implementations
  - Proof of programs properties
  - ...
- Definition of Standard ML - Fully formalized

Conclusion

- The three approaches are complementary
  - operational- how a program is executed
  - denotational- the effect of executing the program
  - axiomatic- axioms saying how to prove properties of a construct
- Different techniques appropriate for different purposes
  - operational when implementing the language
  - denotational when reasoning about programs
  - axiomatic for proving partial correctness of program