Australian National University

COMP3610/6361
Principles of Programming Languages

Peter Höfner

Oct 09, 2023

Section 15

Denotational Semantics

Operational Semantics (Reminder)

- describe how to evaluate programs
- a valid program is interpreted as sequences of steps
- small-step semantics
  - individual steps of a computation
  - more rules (compared to big-step)
  - allows to reason about non-terminating programs, concurrency, . . .
- big-step semantics
  - overall results of the executions
    ‘divide-and-conquer manner’
  - can be seen as relations
  - fewer rules, simpler proofs
  - no non-terminating behaviour
- allow non-determinism

Operational vs Denotational

An *operational semantics* is like an interpreter

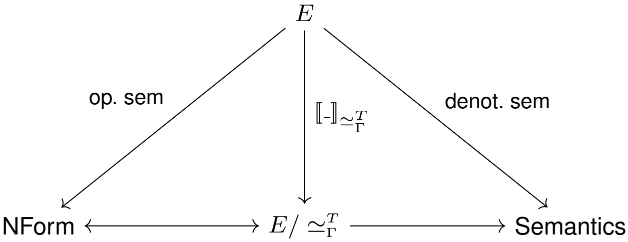
$$\langle E\,,\,s\rangle \longrightarrow \langle E'\,,\,s'\rangle \qquad \text{and} \qquad \langle E\,,\,s\rangle \Downarrow \langle v\,,\,s'\rangle$$

A denotational semantics is like a compiler.
A *denotational semantics* defines what a program means as a (partial) function:

$$\mathcal{C}[\![\texttt{com}]\!] \in \text{Store} \rightharpoonup \text{Store}$$

Allows the use of ‘standard’ mathematics

Big Picture


E
op. sem
$[\![-]\!]_{\simeq_\Gamma^T}$
denot. sem
NForm
$E/\simeq_\Gamma^T$
Semantics

IMP – Syntax (aexp and bexp)

Booleans $b \in \mathbb{B}$
Integers (Values) $n \in \mathbb{Z}$
Locations $l \in \mathbb{L} \ = \{l, l_0, l_1, l_2, \dots\}$

Operations $aop ::= +$

Expressions

$$\text{aexp} ::= n \mid !l \mid \text{aexp}\ aop\ \text{aexp}$$

$$\text{bexp} ::= b \mid \text{bexp} \wedge \text{bexp} \mid \text{aexp} \geq \text{aexp}$$

$$\begin{aligned}\text{com} ::= {} & l := \text{aexp} \mid \\ & \textbf{if}\ \text{bexp}\ \textbf{then}\ \text{com}\ \textbf{else}\ \text{com} \mid \\ & \textbf{skip} \mid \text{com}\ ;\ \text{com} \mid \\ & \textbf{while}\ \text{bexp}\ \textbf{do}\ \text{com}\end{aligned}$$

Semantic Domains

$$\mathcal{C}[\![c]\!] \in \text{Store} \rightharpoonup \text{Store} \qquad \mathcal{C}[\![\_]\!]\_ : \text{com} \to \text{Store} \rightharpoonup \text{Store}$$

$$\mathcal{A}[\![a]\!] \in \text{Store} \rightharpoonup \text{int} \qquad \mathcal{A}[\![\_]\!]\_ : \text{aexp} \to \text{Store} \rightharpoonup \text{int}$$

$$\mathcal{B}[\![b]\!] \in \text{Store} \rightharpoonup \text{bool} \qquad \mathcal{B}[\![\_]\!]\_ : \text{bexp} \to \text{Store} \rightharpoonup \text{bool}$$

Convention: (Partial) Functions are defined point-wise.
$\mathcal{C}[\![\_]\!]$ is the denotation function.

Partial Functions

Remember that partial functions can be represented as sets.

- $\mathcal{C}[\![c]\!]$ can be described as a set
- the equation $\mathcal{C}[\![c]\!] = S$,
  for a set S gives the definition for command c
- $\mathcal{C}[\![c]\!](s)$ is a store

Denotational Semantics for IMP

Arithmetic Expressions

$$\mathcal{A}[\![\underline{n}]\!] = \{(s, n)\}$$

$$\mathcal{A}[\![!l]\!] = \{(s, s(l)) \mid l \in \mathsf{dom}(s)\}$$

$$\mathcal{A}[\![a_1 \underline{+} a_2]\!] = \{(s, n) \mid (s, n_1) \in \mathcal{A}[\![a_1]\!] \wedge (s, n_2) \in \mathcal{A}[\![a_2]\!] \wedge n = n_1 + n_2\}$$

$\underline{n}$ is syntactical, n semantical value.

Denotational Semantics for IMP

Boolean Expressions

$$\mathcal{B}[\![\underline{\texttt{true}}]\!] = \{(s, \texttt{true})\}$$

$$\mathcal{B}[\![\underline{\texttt{false}}]\!] = \{(s, \texttt{false})\}$$

$$\mathcal{B}[\![b_1 \mathbin{\underline{\wedge}} b_2]\!] = \{(s, b) \mid (s, b') \in \mathcal{B}[\![b_1]\!] \wedge (s, b'') \in \mathcal{B}[\![b_2]\!] \wedge (b = b' \wedge b'')\}$$

$$\mathcal{B}[\![a_1 \mathbin{\underline{\geq}} a_2]\!] = \{(s, \texttt{true}) \mid (s, n_1) \in \mathcal{A}[\![a_1]\!] \wedge (s, n_2) \in \mathcal{A}[\![a_2]\!] \wedge n_1 \geq n_2\} \cup \{(s, \texttt{false}) \mid (s, n_1) \in \mathcal{A}[\![a_1]\!] \wedge (s, n_2) \in \mathcal{A}[\![a_2]\!] \wedge n_1 < n_2\}$$

Denotational Semantics for IMP

Arithmetic and Boolean Expressions in Function-Style

$$\mathcal{A}[\![\underline{n}]\!](s) = n$$

$$\mathcal{A}[\![!l]\!](s) = s(l) \quad \text{if } l \in \mathsf{dom}(s)$$

$$\mathcal{A}[\![a_1 \mathbin{\underline{+}} a_2]\!](s) = \mathcal{A}[\![a_1]\!](s) + \mathcal{A}[\![a_2]\!](s)$$

$$\mathcal{B}[\![\underline{\texttt{true}}]\!](s) = \texttt{true}$$

$$\mathcal{B}[\![\underline{\texttt{false}}]\!](s) = \texttt{false}$$

$$\mathcal{B}[\![a_1 \mathbin{\underline{\wedge}} a_2]\!](s) = \mathcal{B}[\![b_1]\!](s) \wedge \mathcal{B}[\![b_2]\!](s)$$

$$\mathcal{B}[\![b_1 \mathbin{\underline{\geq}} a_2]\!](s) = \begin{cases} \texttt{true} & \text{if } \mathcal{A}[\![a_1]\!](s) \geq \mathcal{A}[\![a_2]\!](s) \\ \texttt{false} & \text{otherwise} \end{cases}$$

Denotational Semantics for IMP

Commands

$$\begin{aligned}
\mathcal{C}[\![\textbf{skip}]\!] &= \{(s, s)\} \\
\mathcal{C}[\![l := a]\!] &= \{(s, s + \{l \mapsto n\}) \mid (s, n) \in \mathcal{A}[\![a]\!]\} \\
\mathcal{C}[\![c_1 \ ; \ c_2]\!] &= \{(s, s'') \mid \exists s'.\ (s, s') \in \mathcal{C}[\![c_1]\!] \wedge (s', s'') \in \mathcal{C}[\![c_2]\!]\} \\
\mathcal{C}[\![\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2]\!] &= \{(s, s') \mid (s, \texttt{true}) \in \mathcal{B}[\![b]\!] \wedge (s, s') \in \mathcal{C}[\![c_1]\!]\} \cup \\
&\quad\ \{(s, s') \mid (s, \texttt{false}) \in \mathcal{B}[\![b]\!] \wedge (s, s') \in \mathcal{C}[\![c_2]\!]\}
\end{aligned}$$

Denotational Semantics for IMP

Commands in Function-Style

$$\mathcal{C}[\![\textbf{skip}]\!](s) = s$$

$$\mathcal{C}[\![l := a]\!](s) = s + \{l \mapsto (\mathcal{A}[\![a]\!](s))\}$$

$$\mathcal{C}[\![c_1 \; ; \; c_2]\!] = \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!]$$

$$(\text{or } \mathcal{C}[\![c_1 \; ; \; c_2]\!](s) = \mathcal{C}[\![c_2]\!](\mathcal{C}[\![c_1]\!](s)) \;)$$

$$\mathcal{C}[\![\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2]\!](s) = \begin{cases} \mathcal{C}[\![c_1]\!](s) & \text{if } \mathcal{B}[\![b]\!](s) = \texttt{true} \\ \mathcal{C}[\![c_2]\!](s) & \text{if } \mathcal{B}[\![b]\!](s) = \texttt{false} \end{cases}$$

denotational semantics is often *compositional*

Denotational Semantics for IMP

Commands

(cont'd)

$$\begin{aligned}\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!] = {} & \{(s,s) \mid (s, \texttt{false}) \in \mathcal{B}[\![b]\!]\} \cup \\ & \{(s,s') \mid (s, \texttt{true}) \in \mathcal{B}[\![b]\!] \wedge \\ & \quad \exists s''.\ (s,s'') \in \mathcal{C}[\![c]\!] \wedge (s'',s') \in \mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!](s) &= \mathcal{C}[\![\textbf{if } b \textbf{ then } c\ ;\ (\textbf{while } b \textbf{ do } c) \textbf{ else skip}]\!](s) \\ &= \begin{cases} \mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!](\mathcal{C}[\![c]\!](s)) & \text{if } \mathcal{B}[\![b]\!](s) = \texttt{true} \\ \mathcal{C}[\![\textbf{skip}]\!](s) & \text{if } \mathcal{B}[\![b]\!](s) = \texttt{false} \end{cases}\end{aligned}$$

Problem: this is not a function definition;
it is a recursive equation, we require its solution

Recursive Equations – Example

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ f(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$

Question: What function(s) satisfy this equation?
Answer: $f(x) = x^2$

Recursive Equations – Example II

$$g(x) = g(x) + 1$$

Question: What function(s) satisfy this equation?
Answer: none

Recursive Equations – Example III

$$h(x) = 4 \cdot h\left(\frac{x}{2}\right)$$

Question: What function(s) satisfy this equation?
Answer: multiple

Solving Recursive Equations

Build a solution by approximation (interpret functions as sets)

$$f_0 = \emptyset$$

$$f_1 = \begin{cases} 0 & \text{if } x = 0 \\ f_0(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$
$$= \{(0,0)\}$$

$$f_2 = \begin{cases} 0 & \text{if } x = 0 \\ f_1(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$
$$= \{(0,0), (1,1)\}$$

$$f_3 = \begin{cases} 0 & \text{if } x = 0 \\ f_2(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$
$$= \{(0,0), (1,1), (2,4)\}$$

Solving Recursive Equations

Model this process as higher-order function F that takes the approximation f_k as input and returns the next approximation.

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$

where

$$(F(f))(x) = \begin{cases} 0 & \text{if } x = 0 \\ f(x-1) + 2x - 1 & \text{otherwise} \end{cases}$$

Iterate till a fixed point is reached ($f = F(f)$)

Fixed Point

Definition

Given a function $F : A \to A$, $a \in A$ is a *fixed point* of F if $F(a) = a$.

Notation: Write $a = \text{fix}\,(F)$ to indicate that a is a fixed point of F.

Idea: Compute fixed points iteratively, starting from the completely undefined function. The fixed point is the limit of this process:

$$
\begin{aligned}
f &= \text{fix}\,(F) \\
&= f_0 \cup f_1 \cup f_2 \cup \ldots \\
&= \emptyset \cup F(\emptyset) \cup F(F(\emptyset)) \cup \ldots \\
&= \bigcup_{i \geq 0}^{\infty} F^i(\emptyset)
\end{aligned}
$$

Denotational Semantics for **while**

$$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!] = \textsf{fix}\,(F)$$

where

$$\begin{aligned} F(f) = &\{(s,s) \mid (s, \texttt{false}) \in \mathcal{B}[\![b]\!]\} \cup \\ &\{(s,s') \mid (s, \texttt{true}) \in \mathcal{B}[\![b]\!] \wedge \\ &\qquad \exists s''.\, (s,s'') \in \mathcal{C}[\![c]\!] \wedge (s'',s') \in f\} \end{aligned}$$

Denotational Semantics – Example

$$\mathcal{C}[\![\textbf{while}\ !l \geq 0\ \textbf{do}\ m := !l + !m\ ;\ l := !l + (-1)]\!]$$

$$f_0 = \emptyset$$

$$f_1 = \begin{cases} s & \text{if } !l < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f_2 = \begin{cases} s & \text{if } !l < 0 \\ s + \{l \mapsto -1, m \mapsto s(m)\} & \text{if } !l = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f_3 = \begin{cases} s & \text{if } !l < 0 \\ s + \{l \mapsto -1\} & \text{if } !l = 0 \\ s + \{l \mapsto -1, m \mapsto 1 + s(m)\} & \text{if } !l = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f_4 = \begin{cases} s & \text{if } !l < 0 \\ s + \{l \mapsto -1\} & \text{if } !l = 0 \\ s + \{l \mapsto -1, m \mapsto 1 + s(m)\} & \text{if } !l = 1 \\ s + \{l \mapsto -1, m \mapsto 3 + s(m)\} & \text{if } !l = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Fixed Points

- Why does (fix F) have a solution?
- What if there are several solutions?
(which should we take)

Fixed Point Theory

Definition (sub preserving)

A function F *preserves suprema* if for every chain $X_1 \subseteq X_2 \subseteq \ldots$

$$F(\bigcup_i X_i) = \bigcup_i F(X_i)\ .$$

Lemma

Every suprema-preserving function F is monotone increasing.

$$X \subseteq Y \Longrightarrow F(X) \subseteq F(Y)$$

(works for arbitrary partially ordered sets)

Kleene's fixed point theorem

Theorem

Let F be a suprema-preserving function. The least fixed point of F exists and is equal to

$$\bigcup_{i \geq 0} F^i(\emptyset)$$

$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$

$$
\begin{aligned}
& \mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!](s) \\
= & \ \mathsf{fix}\,(F) \\
= & \begin{cases} \mathcal{C}[\![c]\!]^k(s) & \text{if } k \geq 0 \text{ such that } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]^k(s)) = \texttt{false} \\ & \text{and } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]^i(s)) = \texttt{true} \text{ for all } 0 \leq i < k \\ \text{undefined} & \text{if } \mathcal{B}[\![b]\!](\mathcal{C}[\![c]\!]^i(s)) = \texttt{true} \text{ for all } i \geq 0 \end{cases}
\end{aligned}
$$

This may be what you would have expected, but now it is grounded on well-known mathematics

Exercises

- Show that **skip** $;\, c$ and $c\,;$ **skip** are equivalent.
- What does equivalent mean in the context of denotational semantics?
- Show that $(c_1\ ;\, c_2)\ ;\, c_3$ is equivalent to $c_1\ ;\, (c_2\ ;\, c_3)$.