Australian National University

COMP3610/6361
Principles of Programming Languages

Peter Höfner

Sep 27, 2023

Section 16

Partial and Total Correctness

Styles of semantics

Operational
Meanings for program phrases defined in terms of the steps of computation they can take during program execution.

Denotational
Meanings for program phrases defined abstractly as elements of some suitable mathematical structure.

Axiomatic
Meanings for program phrases defined indirectly via the axioms and rules of some logic of program properties.

Styles of semantics

Operational

– *how* to evaluate programs (interpreter)

– close connection to implementations

Denotational

Meanings for program phrases defined abstractly as elements of some suitable mathematical structure.

Axiomatic

Meanings for program phrases defined indirectly via the axioms and rules of some logic of program properties.

Styles of semantics

Operational

– *how* to evaluate programs (interpreter)
– close connection to implementations

Denotational

– *what* programs calculate (compiler)
– simplifies equational reasoning (semantic equivalence)

Axiomatic

Meanings for program phrases defined indirectly via the axioms and rules of some logic of program properties.

Styles of semantics

Operational
– *how* to evaluate programs (interpreter)
– close connection to implementations

Denotational
– *what* programs calculate (compiler)
– simplifies equational reasoning (semantic equivalence)

Axiomatic
– *describes properties* of programs
– allows reasoning about the correctness of programs

Assertions

Axiomatic semantics *describe properties* of programs. Hence it requires

- a language for expressing properties
- proof rules to establish the validity of properties w.r.t. programs

Examples

- value of l is greater than 0
- value of l is even
- value of l is prime
- eventually the value of l will 0
- $\ldots$

Applications

- proving correctness
- documentation
- test generation
- symbolic execution
- bug finding
- malware detection
- . . .

Assertion Languages

- (English)
- first-order logic ($\forall, \exists, \wedge, \neg, =, R(x), \ldots$)
- temporal and modal logic ($\Box, \Diamond, \bigcirc,$ **Until**, $\ldots$)
- special-purpose logics (Alloy, Z3, . . .)

Assertions as Comments

assertions are (should) be used in code regularly

```
/* Precondition: 0 <= i < A.length */
/* Postcodition: returns A[i] */
public int get (int i) {
        return A[i];
}
```

- useful as documentation or run-time checks
- no guarantee that they are correct
- sometimes not useful (e.g. /*increment i*/)

aim: make this rigorous by defining the semantics of a language using pre- and post-conditions

Partial Correctness

$$\{P\}\ c\ \{Q\}$$

Meaning: if P holds before c, and c executes *and terminates* then Q holds afterwards

Partial Correctness – Examples

- $\{l = 21\}\ l := !l + !l\ \{l = 42\}$
- $\{l = 0 \land m = i\}$
  $k := 0\ ;$
  while $!l \neq\ !m$
  do
  $\quad k := !k - 2\ ;$
  $\quad l := !l + 1$
  $\{k = -i - i\}$

Note: i is a ghost variable
we do not use dereferencing in conditions

Partial Correctness – Examples

The second example is a valid partial correctness statement.

Lemma

$$\forall s, s'. \quad k, l, m \in \textit{\textsf{dom}}(s) \wedge s(l) = 0 \wedge$$
$$\mathcal{C}[\![k := 0 \,;\, \textbf{while } !l \neq !m \textbf{ do } (k := !k - 2 \,;\, l := !l + 1)]\!](s) = s'$$
$$\Longrightarrow s'(k) = -s(m) - s(m)$$

Partial Correctness – Examples

Is the following partial correctness statement valid?

- $\{l = 0 \wedge m = i\}$
  $k := 0\ ;$
  while $!l \neq !m$
  do
  $\quad k := !k + !l\ ;$
  $\quad l := !l + 1$
  $\{k = i\}$

Total Correctness

- partial correctness specifications do not ensure termination
- sometimes termination is needed

$$[P]\ c\ [Q]$$

Meaning: if P holds, then c will terminate and Q holds afterwards

Total Correctness – Example

- $[l = 0 \wedge m = i \wedge \mathbf{i} \geq \mathbf{0}]$
  $k := 0$;
  while $!l \neq !m$
  do
  $\quad k := !k - 2$;
  $\quad l := !l + 1$
  $[k = -i - i]$

Assertions

What properties do we want to state in pre-conditions and post-conditions; so far

- locations (program variables)
- equality
- logical/ghost variables (e.g. i)
- comparison
- we have not used ‘pointers’

choice of assertion language influences the sort of properties we can specify

Assertions – Syntax

Booleans	$b \in \mathbb{B}$
Integers (Values)	$n \in \mathbb{Z}$
Locations	$l \in \mathbb{L} \quad = \{l, l_0, l_1, l_2, \dots\}$
Logical variables	$i \in \mathbf{LVar} = \{i, i_0, i_1, i_2, \dots\}$
Operations	$aop ::= +$

Expressions

$$\mathsf{aexp}_i ::= n \mid l \mid i \mid \mathsf{aexp}_i \; aop \; \mathsf{aexp}_i$$

$$\begin{aligned}\mathsf{assn} ::= {} & b \mid \mathsf{aexp}_i \geq \mathsf{aexp}_i \mid \\ & \mathsf{assn} \wedge \mathsf{assn} \mid \mathsf{assn} \vee \mathsf{assn} \mid \\ & \mathsf{assn} \Rightarrow \mathsf{assn} \mid \neg \mathsf{assn} \mid \\ & \forall i.\ \mathsf{assn} \mid \exists i.\ \mathsf{assn}\end{aligned}$$

Note: bexpincluded in assn; assn not minimal

Assertions – Satisfaction

when does a store s satisfy an assertion

- need interpretation for logical variables

$$I : \mathbf{LVar} \to \mathbb{Z}$$

- denotation function $\mathcal{A}_I[\![\_]\!]$ (similar to $\mathcal{A}[\![\_]\!]$

$$\begin{aligned}
\mathcal{A}_I[\![n]\!](s, I) &= n \\
\mathcal{A}_I[\![l]\!](s, I) &= s(l), \qquad l \in \mathsf{dom}(s) \\
\mathcal{A}_I[\![i]\!](s, I) &= I(i), \qquad i \in \mathsf{dom}(I) \\
\mathcal{A}_I[\![a_1 + a_2]\!](s, I) &= \mathcal{A}_I[\![a_1]\!](s, I) + \mathcal{A}[\![a_2]\!](s, I)
\end{aligned}$$

Assertion Satisfaction

define satisfaction relation for assertions on a given state s

$s \models_I \texttt{true}$	
$s \models_I a_1 \geq a_2$	if $\mathcal{A}_I[\![a_1]\!](s, I) \geq \mathcal{A}_I[\![a_2]\!](s, I)$
$s \models_I P_1 \wedge P_2$	if $s \models_I P_1$ and $s \models_I P_2$
$s \models_I P_1 \vee P_2$	if $s \models_I P_1$ or $s \models_I P_2$
$s \models_I P_1 \Rightarrow P_2$	if $s \not\models_I P_1$ or $s \models_I P_2$
$s \models_I \neg P$	if $s \not\models_I P$
$s \models_I \forall i.\ P$	if $\forall n \in \mathbb{Z}.\ s \models_{I+\{i \mapsto n\}} P$
$s \models_I \exists i.\ P$	if $\exists n \in \mathbb{Z}.\ s \models_{I+\{i \mapsto n\}} P$

an assertion is *valid* ($\models P$) if it is valid in any store, under any interpretation

$$\forall s, I.\ s \models_I P$$

Partial Correctness – Satisfiability

A partial correctness statement $\{P\}\ c\ \{Q\}$ is *satisfied* in store s and under interpretation I ($s \models_I \{P\}\ c\ \{Q\}$) if

$$\forall s'.\ \text{if } s \models_I P \text{ and } \mathcal{C}[\![c]\!](s) = s' \text{ then } s' \models_I Q\ .$$

Partial Correctness – Validity

Assertion validity

An assertion P is *valid* (*holds*) ($\models P$) if it is *valid* in any store under interpretation.

$$\models P \;:\Longleftrightarrow\; \forall s, I.\; s \models_I P$$

Partial correctness validity

A partial correctness statement $\{P\}\; c\; \{Q\}$ is *valid* ($\models \{P\}\; c\; \{Q\}$) if it is valid in any store under interpretation.

$$\models \{P\}\; c\; \{Q\} \;:\Longleftrightarrow\; \forall s, I.\; s \models_I \{P\}\; c\; \{Q\}$$

Proving Specifications

how to proof the (partial) correctness of $\{P\}\ c\ \{Q\}$

- show $\forall s, I.s \models_I \{P\}\ c\ \{Q\}$
- $s \models_I \{P\}\ c\ \{Q\}$ requires denotational semantics $\mathcal{C}$

- we can do this manually, but ...
- we can derive inference rules and axioms (axiomatic semantics)
- allows derivation of correctness statements without reasoning about stores and interpretations