CS6848 - Principles of Programming Languages

Principles of Programming Languages

V. Krishna Nandivada

IIT Madras

Axiomatic semantics

- Operational semantics talks about how an expression is evaluated.
- Denotational semantics - describes what a program text means in mathematical terms - constructs mathematical objects.
- Axiomatic semantics - describes the meaning of programs in terms of properties (axioms) about them.
- Usually consists of
  - A language for making assertions about programs.
  - Rules for establishing when assertions hold for different programming constructs.

Language for Assertions

- A specification language
  - Must be easy to use and expressive
  - Must have syntax and semantics.
- Requirements:
  - Assertions that characterize the state of execution.
  - Refer to variables, memory
- Examples of non state based assertions:
  - Variable x is live,
  - Lock L will be released.
  - No dependence between the values of x and y.

Assertion Language

- Specification language in first-order predicate logic
  - Terms (variables, constants, arithmetic operations)
  - Formulas:
    - `true` and `false`
    - If t_1 and t_2 are terms then, $t_1 = t_2$, $t_1 < t_2$ are formulas.
    - If ϕ is a formula, so is $\neg\phi$.
    - IF ϕ_1 and ϕ_2 are two formulas then so are $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\phi_1 \Rightarrow \phi_2$.
    - If $\phi(x)$ is a formula (with a free variable x) then, $\forall x.\phi(x)$ and $\exists x.\phi(x)$ are formulas.

Hoare Triples

- Meaning of a statement S can be described in terms of triples:
  $\{P\}S\{Q\}$
  where
- P and Q are formulas or assertions.
  - P is **a** pre-condition on S
  - Q is **a** post-condition on S.
- The triple is *valid* if
  - execution of S begins in a state satisfying P.
  - S terminates.
  - resulting state satisfies Q.

Satisfiability

- A formula in first-order logic can be used to characterize states.
  - The formula $x = 3$ characterizes all program states in which the value of the location associated with x is 3.
  - Formulas can be thought as assertions about states.
- Define $\{\sigma \in \Sigma | \sigma \models \phi\}$, where $\models$ is a satisfiability relation.
  - Let the value of a term t in state σ be t^σ
    - If t is a variable x then $t^\sigma = \sigma(x)$.
    - If t is an integer n then $t^\sigma = n$.
    - $\sigma \models t_1 = t_2$ if $t_1^\sigma = t_2^\sigma$
    - $\sigma \models t_1 \wedge t_2$ if $\sigma \models t_1$ and $\sigma \models t_2$
    - $\sigma \models \forall x.\phi(x)$ if $\sigma[x \mapsto n] \models \phi(n)$ for all integer constants n.
    - $\sigma \models \exists x.\phi(x)$ if $\sigma[x \mapsto n] \models \phi(n)$ for some integer constant n.

Examples

- $\{2 = 2\}x := 2\{x = 2\}$
  An assignment operation of x to 2 results in a state in which x is 2, assuming equality of integers!
- $\{\texttt{true}\}$ `if` B then $x := 2$ `else` $x := 1$ $\{x = 1 \vee x = 2\}$
  A conditional expression that either assigns x to 1 or 2, if executed will lead to a state in which x is either 1 or 2.
- $\{2 = 2\}x := 2\{y = 1\}$
- $\{\texttt{true}\}$ `if` B then $x := 2$ `else` $x := 1$ $\{x = 1 \wedge x = 2\}$
  Why are these invalid?

Partial Correctness

- The validity of a Hoare triple depends upon the termination of the statement S
- $\{0 \leq a \wedge 0 \leq b\}\ S\ \{z = a \times b\}$
  - If executed in a state in which $0 \leq a$ and $0 \leq b$, and
  - S terminates,
  - then $z = a \times b$.

Soundness

- Hoare rules can be seen as a proof system.
  - Derivations are proofs.
  - conclusions are theorems.
  - We write $\vdash$ {P} c {Q}, if {P} c {Q} is a theorem.
- If $\vdash$ {P} c {Q}, then $\models$ {P} c {Q}.
  - Any derivable assertion is *sound* with respect to the underlying semantics.

Proof rules

- Skip:
  $$\{P\} skip \{P\}$$
- Assignment:
  $$\{P[t/x]\} x := t \{P\}$$
  Example: Suppose $t = x+1$
  then, $\{x+1=2\} x := x+1 \{x=2\}$
- $$\text{Sequencing}\ \frac{\{P_1\}c_0\{P_2\}\ \{P_2\}c_1\{P_3\}}{\{P_1\}c_0;c_1\{P_3\}}$$
- $$\text{Conditionals}\ \frac{\{P_1 \wedge b\}c_0\{P_2\}\ \{P_1 \wedge \neg b\}c_1\{P_2\}}{\{P_1\}\texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1\{P_2\}}$$

Proof rules (contd)

- $$\text{Loop}\ \frac{\{P \wedge b\}c\{P\}}{\{P\}\texttt{while}\ b\ c\{P \wedge \neg b\}}$$
- $$\text{Consequence}\ \frac{\models (P \Rightarrow P'), \{P'\}c\{Q'\}, \models (Q' \Rightarrow Q)}{\{P\}c\{Q\}}$$
  strengthening of P' to P, and weakening of Q' to Q.

Examples

- $\{x > 0\}\ y = x-1\ \{y \geq 0\}$ implies
  $\{x > 10\}\ y = x-1\ \{y \geq -5\}$
- $\{x > 0\}\ y = x-1\ \{y \geq 0\}$ and
  $\{y \geq 0\}\ x = y\ \{x \geq 0\}$ implies
  $\{x > 0\}\ y = x-1; x = y\ \{x \geq 0\}$

Apply rules of consequence to arrive at universal pre-condition and post-condition

Use of Axiomatic semantics to properties

Prove that the following program:

```
z := 0;
n := y;

while n > 0 do
  z := z + x;
  n := n - 1;
```

computes the product of `x` and `y` (assuming y is non-negative).

Step I - choosing the invariants

- Want to show the following Hoare triple is valid:
  $\{y \geq 0\}$ *above-program* $\{z = x * y\}$
- Invariant for the `while` loop:
  $P = \{z = x*(y-n) \wedge n \geq 0\}$

Step II - constructing the proof in reverse order

```
{z = x * (y-n) ∧ n ≥ 0}
while n > 0 do z := z+x; n := n-1
{z = x * y}

z = x * (y-n) ∧ n ≥ 0 ∧ ¬ (n > 0) ⇒ z = x * y
(definition of while)

(apply the consequence rule)
{z = x * (y-n) ∧ n ≥ 0}
while n > 0 do z := z+x; n := n-1
{z = x * (y-n) ∧ n ≥ 0 ∧ ¬ (n > 0) }
```


Step II - constructing the proof in reverse order

```
(any iteration)
{(z+x) = x * (y-(n-1)) ∧ (n-1) ≥ 0}
z := z+x;
{z=x*(y-(n-1)) ∧ (n-1) ≥ 0}
n := n-1
{z=x*(y-n) ∧ n ≥ 0}

z = x*(y-n) ∧ n ≥ 0 ∧ n > 0 ⇒
        {(z+x) = x * (y-(n-1)) ∧ (n-1) ≥ 0}

(consequence)
{z = x*(y-n) ∧ n ≥ 0 ∧ n > 0}
z := z+x; n := n-1
{z=x*(y-n) ∧ n ≥ 0}
```

Step II - constructing the proof in reverse order

```
(pre-loop code)
{z = x*(y-y) ∧ y ≥ 0}
n := y
{z = x*(y-n) ∧ n ≥ 0}

{0 = x*(y-y) ∧ y ≥ 0}
z := 0
{z = x*(y-y) ∧ y ≥ 0}


{y ≥ 0}
z := 0; n := y
{z = x*(y-n) ∧ n ≥ 0}
{y ≥ 0} above-program {z = x * y}
```


Useless assignment

```
while (x != y) do
if (x <= y)
then
y := y-x
else
x := x-y
```

Derive that
⊢ {x = m ∧ y = n} above-program {x = gcd(m, n)}

Hint: Start with the loop invariant to be {gcd(x, y) = gcd(m, n)}

Last Class

- Axiomatic Semantics
- Proof rules
- Proving the semantics of the multiplication routine.

More proofs

- Proving that the factorial (using loops) computes factorial
- Proving that the exp (using loops) computes exp (M, N).

Connection between axiomatic and operational semantics

- Semantics of Valid assertions
- Soundness
- Completeness

Validity

Validity via Partial correctness

- $\{P\}c\{Q\}$: Whenever we start the execution of command c in a state that satisfies P, the program either does not terminate or it terminates in a state that satisfies Q.
- $\forall \sigma, P, Q, c \models \{P\}c\{Q\}$
  if
  $\forall \sigma'$:
  $\sigma \triangleright P \vdash \langle true, \sigma \rangle \wedge$
  $\sigma \triangleright c \vdash \sigma'$
  then
  $\sigma' \triangleright Q \vdash \langle true, \sigma' \rangle$

Validity

Validity via total correctness

- $[P]c[Q]$: Whenever we start the execution of command c in a state that satisfies P, the program terminates in a state that satisfies Q.
- $\forall \sigma, P, Q, c \models [P]c[Q]$
  if $\sigma \triangleright P \vdash \langle true, \sigma \rangle$
  then
  $\exists \sigma'$:
  $\sigma \triangleright c \vdash \sigma' \wedge$
  $\sigma' \triangleright Q \vdash \langle true, \sigma' \rangle$
- note the square brackets (not curly brackets).

Derivations and Validity

- We use $\vdash A$ to indicate that we can prove (derive) the assertion A.
- We use $\vdash \{A\}c\{B\}$ to indicate that we can prove the partial correctness assertion $\{A\}c\{B\}$.
- We wish that $\models \{A\}c\{B\}$ iff $\vdash \{A\}c\{B\}$.

Soundness

- All derived triples are valid.
- If $\vdash$ {P} c {Q}, then $\models$ {P} c {Q}.
  - Any derivable assertion is *sound* with respect to the underlying operational semantics.
- Soundness is guaranteed from our proof rules.

Completeness

- All derived triples are derivable from empty set of assumptions.
- If $\models$ {P} c {Q}, then
  $\exists\sigma'$
  $\quad$ $\textit{init-state} \ \triangleright \{P\}c\{Q\} \vdash \langle \textit{true}, \sigma' \rangle$.
- Harder to achieve (in general) – complete only if the underlying logic is complete if ($\models A$) then $\vdash A$.

Acknowledgements

- Suresh Jagannathan
- George Necula
- Internet.