Review

- Operational semantics
  - relatively simple
  - many flavors (small vs. big)
  - not compositional (rule for while)
- Good for
  - describing language implementation
  - reasoning about properties of the language
    - eg. determinism
  - reasoning about of tools that manipulate programs
    - eg. compilers, type checkers, etc

Not good for automatic reasoning about programs

Axiomatic Semantics

1. Language for making assertions about programs
2. Rules for establishing, i.e. proving the assertions

Typical kinds of assertions:

- This program terminates.
- During execution if var **z** has value 0, then **x** equals **y**
- All array accesses are within array bounds

Some typical languages of assertions:

- First-order logic
- Other logics (e.g., temporal logic)

Axiomatic Semantics

History: Program Verification

- Turing 1949: Checking a large routine
- Floyd 1967: Assigning meaning to programs
- Hoare 1971: An "axiomatic basis for computer programming"
- Led to axiomatic semantics for substantial languages

- Program Verifiers (70's – 80's)
- PREfix: Symbolic Execution for bug-hunting (WinXP)
- Software Validation tools
- PCC
- Malware detection
- Automatic Test Generation …

Critical for algorithmic software analysis

Dijkstra Said

"Program testing can be used to show the presence of bugs, but never to show their absence!"

Hoare Said

- Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, reliability, documentation, and compatibility. However, program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs.

C.A.R Hoare,
"An Axiomatic Basis for Computer Programming",
1969

What happened ?

- The project of defining and proving everything formally has not succeeded (at least not yet)
- Proving has not replaced testing (and praying)
- Machines are getting faster
- Ambitions are getting smaller
- Techniques are getting cooler
- So, pay attention to the next three lectures…

Assertions for IMP

- Partial correctness assertion: {A} **c** {B}

  If A holds in state σ *and there exists* σ' *s.t.* $\langle \mathbf{c}, \sigma \rangle \Downarrow \sigma'$ *then B holds in* σ'

- Total correctness assertion: [A] **c** [B]

  If A holds in state σ *then there exists* σ' *s.t.* $\langle \mathbf{c}, \sigma \rangle \Downarrow \sigma'$ *and B holds in* σ'

- These are called Hoare triples
  - A is called precondition and B is called postcondition
- Example: $\{ y \leq x \}$ `z := x; z := z +1` $\{ y < z \}$

The Assertion Language

- Arith Exprs + First-order predicate logic

$$
\begin{aligned}
A ::= \ & \text{true} \mid \text{false} \\
& \mid e_1 = e_2 \mid e_1 \geq e_2 \\
& \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \Rightarrow A_2 \\
& \mid \forall x.A \mid \exists x.A
\end{aligned}
$$

- IMP boolean expressions are assertions

Semantics of Assertions

Judgement $\sigma \models A$ means assertion holds in given state

$\sigma \models \text{true}$	always
$\sigma \models e_1 = e_2$	iff $\langle e_1, \sigma\rangle \Downarrow n_1$ and $\langle e_2, \sigma\rangle \Downarrow n_2$ and $n_1 = n_2$
$\sigma \models e_1 \geq e_2$	iff $\langle e_1, \sigma\rangle \Downarrow n_1$ and $\langle e_2, \sigma\rangle \Downarrow n_2$ and $n_1 \geq n_2$
$\sigma \models A_1 \wedge A_2$	iff $\sigma \models A_1$ and $\sigma \models A_2$
$\sigma \models A_1 \vee A_2$	iff $\sigma \models A_1$ or $\sigma \models A_2$
$\sigma \models A_1 \Rightarrow A_2$	iff $\sigma \models A_1$ implies $\sigma \models A_2$
$\sigma \models \forall x.A$	iff $\forall n \in \mathbb{Z}.\sigma[x \mapsto n] \models A$
$\sigma \models \exists x.A$	iff $\exists n \in \mathbb{Z}.\sigma[x \mapsto n] \models A$

Semantics of Assertions

Formal definition of partial correctness assertion:

$\models \{ A \}\ c\ \{ B \}$ holds if and only if

$\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma.\ \langle c, \sigma\rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$

... and total correctness assertion:

$\models [A]\ c\ [B]$ holds if and only if

$\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow [\exists \sigma' \in \Sigma.\ \langle c, \sigma\rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$

$\wedge\ \ \forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow [\exists \sigma' \in \Sigma.\ \langle c, \sigma\rangle \Downarrow \sigma']$

Deriving Assertions

- Formal definition of $\models \{A\}\ \mathbf{c}\ \{B\}$ is hard to use

  $\models \{A\}\ \mathbf{c}\ \{B\}$ holds if and only if

  $\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma.\ \langle c, \sigma\rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$

  – defined in terms of the operational semantics

- Next, a symbolic technique (a logic) for deriving valid triples $\vdash \{A\}\ \mathbf{c}\ \{B\}$ from other valid triples

Derivation Rules for Hoare Triples

- We write $\vdash \{A\}\ \mathbf{c}\ \{B\}$ when we can derive the triple using derivation rules
- One derivation rule for each command
- Plus, the rule of consequence:

$$\frac{A' \Rightarrow A \qquad \vdash \{A\}\ \mathbf{c}\ \{B\} \qquad B \Rightarrow B'}{\vdash \{A'\}\ \mathbf{c}\ \{B'\}}$$

Deriv. Rules for Hoare Logic $\vdash$ **{A} c {B}**

Rules for each language construct

$$\frac{}{\vdash \{A\}\ \mathbf{skip}\ \{A\}} \qquad \frac{\vdash \{A\}\ \mathbf{c_1}\ \{B\} \quad \vdash \{B\}\ \mathbf{c_2}\ \{C\}}{\vdash \{A\}\ \mathbf{c_1;\ c_2}\ \{C\}}$$

$$\frac{\vdash \{A \wedge b\}\ \mathbf{c_1}\ \{B\} \quad \vdash \{A \wedge \neg b\}\ \mathbf{c_2}\ \{B\}}{\vdash \{A\}\ \mathbf{if\ b\ then\ c_1\ else\ c_2}\ \{B\}}$$

$$\frac{\vdash \{A \wedge b\}\ \mathbf{c}\ \{A\}}{\vdash \{A\}\ \mathbf{while\ b\ do\ c}\ \{A \wedge \neg b\}} \qquad \frac{}{\vdash \{[e/x]A\}\ \mathbf{x:=e}\ \{A\}}$$

And the rule of consequence…

Free and Bound Variables

Key idea in logic/PL: scoping & substitution

- Assertions are equivalent up to renaming of bound variables (a.k.a. alpha-renaming)
- Examples:

  $\forall x.x = x$ is the same as $\forall y.y = y$

  – Rename bound x with y

  $\forall x.\ \forall y.x = y$ is the same as $\forall z.\ \forall x.z = x$

  – Rename bound x with z and y with x

Substitution

- $[e'/x]\ e$ is substituting e' for x in e
  - Also written as $e[e'/x]$
  - Note: only substitute the free occurrences
- Alpha-rename bound variables to avoid conflicts
  - To subst. $[e'/x]$ in $\forall y.x = y$ rename y if it occurs in e'
  - Result of alpha-renaming: $\forall z.\ e' = z$
- We say that substitution avoids variable capture

  $[\ x/z\ ]\ \forall x.z = x$ is ?
  - $\forall x.x = x$ Wrong
  - $\forall y.x = y$ Correct

Other Hoare Rules

- Multiple rules possible for some constructs

$$\frac{}{\vdash \{A\}\ \mathbf{x:=e}\ \{\exists x_0.[x_0/x]A \wedge x=[x_0/x]e\}}$$

$$\frac{\vdash A \wedge b \Rightarrow I \quad \vdash \{I\}\ \mathbf{c}\ \{A\} \quad \vdash A \wedge \neg b \Rightarrow B}{\vdash \{A\}\ \mathbf{while\ b\ do\ c}\ \{B\}}$$

- Exercise: Derive new rules from old ones using the rule of consequence

Example: Assignment

Prove $\{true\}$ **x:=e** $\{ x = e \}$

Example: Assignment

Assume **x** does not appear in **e**

Prove $\{true\}$ **x:=e** $\{ x = e \}$

As $[e/x](x = e) \equiv e = [e/x]e \equiv e = e$

Use assignment rule ...

Then use consequence rule

$$\frac{true \Rightarrow e = e \qquad \dfrac{}{\vdash \{e = e\}\ \mathbf{x:=e}\ \{x = e\}}}{\vdash \{true\}\ \mathbf{x:=e}\ \{x = e\}}$$

Assignment Axiom and Aliasing

Does assignment rule work with aliasing ?

If **x** and **y** are aliased then:

$\{true\}$ **x:=5** $\{x+y=10\}$

More on this later ...

Example: Conditional

Prove: $\{true\}$ **if y≤0 then x:=1 else x:=y** $\{x>0\}$

$$\dfrac{\dfrac{true \wedge y \le 0 \Rightarrow 1>0 \quad \vdash \{1>0\}\ \mathbf{x:=1}\ \{x>0\}}{\vdash \{true \wedge y \le 0\}\ \mathbf{x:=1}\ \{x>0\}} \qquad \dfrac{true \wedge y>0 \Rightarrow y>0 \quad \vdash \{y>0\}\ \mathbf{x:=y}\{x>0\}}{\vdash \{true \wedge y>0\}\ \mathbf{x:=y}\ \{x>0\}}}{\vdash \{true\}\ \mathbf{if\ y \le 0\ then\ x:=1\ else\ x:=y}\ \{x>0\}}$$

- Rule for if-then-else
- Rule for assignment + consequence

Example: Loop

- Prove: $\vdash \{x \le 0\}$ **while x≤5 do x:=x+1** $\{x=6\}$
- Use the rule for while with invariant $x \le 6$:

$$\dfrac{\dfrac{x \le 6 \wedge x \le 5 \Rightarrow x+1 \le 6 \qquad \dfrac{}{\vdash \{x+1 \le 6\}\ \mathbf{x:=x+1}\ \{x \le 6\}}}{\vdash \{x \le 6 \wedge x \le 5\}\ \mathbf{x:=x+1}\ \{x \le 6\}}}{\vdash \{x \le 6\}\ \mathbf{while\ x \le 5\ do\ x:=x+1}\ \{x \le 6 \wedge x>5\}}$$

- Finish off with consequence rule:

$$\dfrac{x \le 0 \Rightarrow x \le 6 \qquad \vdash \{x \le 6\}\ \mathbf{while\ldots}\ \{x \le 6 \wedge x>5\} \qquad x \le 6 \wedge x>5 \Rightarrow x=6}{\vdash \{x \le 0\}\ \mathbf{while\ldots}\ \{x = 6\}}$$

Another Example

Prove that $\vdash \{A\}$ **while true do c** $\{B\}$

holds for any A, B, and **c**

Proof: construct a derivation tree

$$\dfrac{A \Rightarrow true \quad \dfrac{\vdash \{true \wedge true\}\ \mathbf{c}\ \{true\}}{\{true\}\mathbf{while\ true\ do\ c}\{true \wedge false\}} \quad true \wedge false \Rightarrow B}{\vdash \{A\}\ \mathbf{while\ true\ do\ c}\ \{B\}}$$

Need a lemma: $\forall A \forall c \vdash \{A\}\ \mathbf{c}\ \{true\}$

Notes on Using Hoare Rules

Mostly syntax directed, but:

- When to apply the rule of consequence ?
- What invariant to use for while ?
- How to prove implications (in conseq rule)?

The last one involves theorem proving:

- Doable
- Loop invariants are the hardest problem

Soundness of Axiomatic Semantics

Formal statement of soundness:

If $\vdash$ {A} **c** {B} then $\vDash$ {A} **c** {B}

or, equivalently

If H:: $\vdash$ {A} **c** {B} then

for all σ s.t. $\sigma \vDash A$ and D::<**c**,σ> $\Downarrow$ σ'

we have $\sigma' \vDash B$

Proof: simultaneous ind. on structure of D and H

Hoare Rules: Assignment and References

- When is the following Hoare triple valid?

  { A } ***x := 5** { *x + *y = 10 }

- A should be " *y = 5 or x = y "
- but Hoare rule for assignment gives:

  [5/*x](*x + *y = 10)

  = 5 + *y = 10

  = *y = 5

  (uh oh! we lost one case! What gives ?)

Hoare Rules: Assignment and References

Modeling writes with memory expressions

- Treat memory as a whole w/ memory variables (M)
- upd(M,E_1,E_2) : update M at addr E_1 with value E_2
- sel(M,E_1) : read M at address E_1

Reason about memory expressions with McCarthy's rule

$$\text{sel}(\text{upd}(M, E_1, E_2), E_3) = \begin{cases} E_2 & \text{if } E_1 = E_3 \\ \text{sel}(M, E_3) & \text{if } E_1 \neq E_3 \end{cases}$$

Assignment (update) changes the value of memory

$$\frac{}{\{B[\text{upd}(M, E_1, E_2)/M]\}\ *\mathbf{E_1} := \mathbf{E_2}\ \{B\}}$$

Memory Aliasing

- Consider again: {A} ***x:=5** {*x+*y=10 }
- We obtain:

  A = [upd(M, x, 5)/M] (*x+*y=10)

  = [upd(M, x, 5)/M] (sel(M,x) + sel(M,y) = 10)

  = sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10

  = 5 + sel(upd(M, x, 5), y) = 10

  = if x = y then 5 + 5 = 10 else 5 + sel(M, y) = 10

  = x=y or *y = 5

Next time…

Algorithmic analysis w/ axiomatic semantics

- Weakest Preconditions
- Strongest Postconditions