Announcements

- Homework 5 is due tonight
- No take-home midterm

Questions?

Review

Notation: Assertions

\{ A \} \subseteq \{ B \}

with the meaning that:
- if A holds in state \( \sigma \) and if \( \langle c, \sigma \rangle \Downarrow \sigma' \)
- then B holds in \( \sigma' \)

- A is the precondition
- B is the postcondition
- For example:
  \( \{ y \leq x \} \ z := x; \ z := z + 1 \{ y < z \} \)
  is a valid assertion
- These are called Hoare triples or Hoare assertions

Semantics of Assertions: \( \vdash \)

- Relation \( \sigma \vdash A \) to say that an assertion holds in a given state (= "A is true in \( \sigma \)"")

- The \( \vdash \) relation is defined inductively on the structure of assertions
Derivation Rules

- We write \( \vdash A \) when \( A \) can be derived from basic axioms (\( \vdash A \iff \text{"we can prove } A \" )

- The derivation rules for the judgment \( \vdash A \) are the usual ones from first-order logic with arithmetic

Propositional Rules

Natural deduction style

First-Order Rules

- substitution: "substitute \( a \) for \( x \) in \( A \"

Derivation Rules for Hoare Triples

- Similarly we write \( \vdash \{ A \} c \{ B \} \) when we can derive the triple using derivation rules

- There is one derivation rule for each command in the language

- Plus, the "evil" rule of consequence

Derivation Rules for Hoare Logic

- One rule for each syntactic construct:
Derivation Rules for Hoare Logic

- One rule for each syntactic construct:

\[\vdash (A) \text{skip} \{A\} \quad \vdash ([e/x]A) x := e \{A\}\]
\[\vdash (A) c_1 (B) \quad \vdash (B) c_2 (C)\]
\[\vdash (A) c_1 c_2 (C)\]
\[\vdash (A \land b) c_1 (B) \quad \vdash (A \land \neg b) c_2 (B)\]
\[\vdash (A) \text{if } b \text{ then } c_1 \text{ else } c_2 (B)\]
\[\vdash (A \land b) c (A)\]
\[\vdash (A) \text{while } b \text{ do } c (A \land \neg b)\]

Alternate Hoare Rules

- For some constructs multiple rules are possible:
  (Exercise: these rules can be derived from the previous ones using the consequence rules)

\[\vdash (A) x := e \{\exists_{x/}[x/x]A \land x = [x_0/x]e\}\]
(This one is called the "forward" axiom for assignment)
\[\vdash A \land b \Rightarrow C \quad \vdash (C) c (A) \quad \vdash A \land \neg b \Rightarrow B\]
\[\vdash (A) \text{while } b \text{ do } c (B)\]
(C is the loop invariant)

Example Verifications

Example: Assignment

- (Assuming that \(x\) does not appear in \(e\))
  Prove that \(\{\text{true}\} x := e \{x = e\}\)

\[
\vdash \text{true} \Rightarrow \text{true} \quad \vdash \exists_{x_0} \exists_{x_0} x_0 = e \quad \vdash x_0 \Rightarrow x_0 = e
\]
\[\vdash \{\text{true}\} x := e \{x = e\}\]

Example: Assignment

- (Assuming that \(x\) does not appear in \(e\))
  Prove that \(\{\text{true}\} x := e \{x = e\}\)

\[
\vdash \{e = e\} x := e \{x = e\}
\]
because \([e/x](x = e) \Rightarrow e = e\)

- Use Assignment + Consequence:
  \[\vdash \text{true} \Rightarrow e = e\]
  \[\vdash \{e = e\} x := e \{x = e\}\]
\[\vdash \{\text{true}\} x := e \{x = e\}\]
The Assignment Axiom

- "Assignment is undoubtedly the most characteristic feature of programming a digital computer, and one that most clearly distinguishes it from other branches of mathematics. It is surprising therefore that the axiom governing our reasoning about assignment is quite as simple as any to be found in elementary logic." - Tony Hoare

- Caveats are sometimes needed for languages with aliasing (the strong update problem):
  - If \( x \) and \( y \) are aliased then
    - \( \text{true} \lor x := 5 \ (x 
eq y = 10) \)

Example: Conditional

\[ D_1 :: \vdash (\text{true} \land y \leq 0) \ x := 1 \ (x > 0) \]
\[ D_2 :: \vdash (\text{true} \land y > 0) \ x := y \ (x > 0) \]
\[ \vdash (\text{true}) \text{ if } y \leq 0 \text{ then } x := 1 \text{ else } x := y \ (x > 0) \]

- \( D_1 \) and \( D_2 \) were obtained by consequence and assignment. \( D_1 \) details:

\[ \vdash (1 > 0) \ x := 1 \ (x > 0) \]
\[ \vdash \text{true} \land y \leq 0 \Rightarrow 1 > 0 \]
\[ \vdash (\text{false} \land y \leq 0) \ x := 1 \ (x > 0) \]

Example: Loop

- We want to derive that
  \[ \vdash \{ x \leq 0 \} \text{ while } x \leq 5 \ x := x + 1 \ (x = 6) \]

- Use the rule for while with invariant \( A = ? \)

\[ \vdash (A) \text{ while } x \leq 5 \ x := x + 1 \ (A \land x > 5) \]

- Then finish-off with consequence

\[ \vdash \{ x \leq 0 \} \text{ while } \ldots \ (x = 6) \]

Using Hoare Rules

- Hoare rules are mostly syntax directed

- There are three wrinkles:
  - What invariant to use for while? (fixpoints, widening)
  - When to apply consequence? (theorem proving)
  - How do you prove the implications involved in consequence? (theorem proving)

- This is how theorem proving gets in the picture
  - This turns out to be doable!
  - The loop invariants turn out to be the hardest problem!

(Should the programmer give them? See Dijkstra, ESC)
Where Do We Stand?

- We have a language for asserting properties of programs
- We know when such an assertion is true
- We also have a symbolic method for deriving assertions

\[ \sigma \models A \]
\[ \vdash (A) \subseteq (B) \]