

Axiomatic Semantics

Programs → Theorems. Axiomatic Semantics

- Consists of:
 - A language for making assertions about programs
 - Rules for establishing when assertions hold
- Typical assertions:
 - During the execution, only non-null pointers are dereferenced
 - This program terminates with $x = 0$
- Partial vs. total correctness assertions
 - Safety vs. liveness properties
 - Usually focus on safety (partial correctness)

Partial Correctness Assertions

- The assertions we make about programs are of the form:

$$\{A\} c \{B\}$$

with the meaning that:

- Whenever we start the execution of c in a state that satisfies A , the program either does not terminate or it terminates in a state that satisfies B
- A is called precondition and B is called 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

Total Correctness Assertions

- $\{A\} c \{B\}$ is a partial correctness assertion. It does not imply termination
- $[A] c [B]$ is a total correctness assertion meaning that
Whenever we start the execution of c in a state that satisfies A the program does terminate in a state that satisfies B
- Now let's be more formal
 - Formalize the language of assertions, A and B
 - Say when an assertion holds in a state
 - Give rules for deriving Hoare triples

Languages for Assertions

- A specification language
 - Must be easy to use and expressive (conflicting needs)
 - Most often only expression \odot
 - Syntax: how to construct assertions
 - Semantics: what assertions mean
- Typical examples
 - First-order logic
 - Temporal logic (used in protocol specification, hardware specification)
 - Special-purpose languages: Z, Larch, Java ML

State-Based Assertions

- Assertions that characterize the state of the execution
 - Recall: state = state of locals + state of memory
- Our assertions will need to be able to refer to
 - Variables
 - Contents of memory
- What are not state-based assertions
 - Variable x is live, lock L will be released
 - There is no correlation between the values of x and y

An Assertion Language

- We'll use a fragment of first-order logic first
 - Formulas $A ::= O \mid T \mid \perp \mid P_1 \wedge P_2 \mid \forall x.P \mid P_1 \Rightarrow P_2 \mid$
 - Atoms $O ::= f(O_1, \dots, O_n) \mid E_1 \leq E_2 \mid E_1 = E_2 \mid \dots$
 - Exprs $E ::= n \mid \text{true} \mid \text{false} \mid \dots$
- We can also have an arbitrary assortment of function symbols
 - $\text{ptr}(E, t)$ - expression E denotes a pointer to a t
 - $E : \text{ptr}(t)$ - same in a different notation
 - $\text{reachable}(E_1, E_2)$ - list cell E_2 is reachable from E_1
 - these can be built-in or defined

Automated Deduction - George Necula - Lecture 2

7

Semantics of Assertions

- We introduced a language of assertions, we need to assign meanings to assertions.
 - We ignore for now references to memory
- Notation $\rho, \sigma \models A$ to say that an assertion holds in a given state.
 - This is well-defined when ρ is defined on all variables occurring in A and σ is defined on all memory addresses referenced in A
- The \models judgment is defined inductively on the structure of assertions.

Automated Deduction - George Necula - Lecture 2

8

Semantics of Assertions

- Formal definition (we drop σ for simplicity):

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

Automated Deduction - George Necula - Lecture 2

9

Semantics of Assertions

- Now we can define formally the meaning of a partial correctness assertion
 - $\models \{A\} c \{B\}$:
 $\forall \rho \sigma. \forall \rho' \sigma'. (\rho, \sigma \models A \wedge \rho, \sigma \vdash c \Downarrow \rho', \sigma') \Rightarrow \rho', \sigma' \models B$
- ... and the meaning of a total correctness assertion
 - $\models [A] c [B]$ iff
 $\forall \rho \sigma. \forall \rho' \sigma'. (\rho, \sigma \models A \wedge \rho, \sigma \vdash c \Downarrow \rho', \sigma') \Rightarrow \rho', \sigma' \models B$
 \wedge
 $\forall \rho \sigma. \rho, \sigma \models A \Rightarrow \exists \rho' \sigma'. \rho, \sigma \vdash c \Downarrow \rho', \sigma'$

Automated Deduction - George Necula - Lecture 2

10

Why Isn't This Enough?

- Now we have the formal mechanism to decide when $\{A\} c \{B\}$
 - Start the program in all states that satisfies A
 - Run the program
 - Check that each final state satisfies B
- This is exhaustive testing
- Not enough
 - Can't try the program in all states satisfying the precondition
 - Can't find all final states for non-deterministic programs
 - And also it is impossible to effectively verify the truth of a $\forall x.A$ postcondition (by using the definition of validity)

Automated Deduction - George Necula - Lecture 2

11

Derivations as Proxies for Validity

- We define a symbolic technique for deriving valid assertions from others that are known to be valid
 - We start with validity of first-order formulas
- We write $\vdash A$ when we can derive (prove) assertion A
 - We wish that $(\forall \rho \sigma. \rho, \sigma \models A)$ iff $\vdash A$
- We write $\vdash \{A\} c \{B\}$ when we can derive (prove) the partial correctness assertion
 - We wish that $\models \{A\} c \{B\}$ iff $\vdash \{A\} c \{B\}$

Automated Deduction - George Necula - Lecture 2

12

Derivation Rules for Assertions

- The derivation rules for $\vdash A$ are the usual ones from first-order logic with
- Natural deduction style axioms:

$$\frac{}{\vdash A \wedge B} \quad \frac{}{\vdash [a/x]A \text{ (a is fresh)}} \quad \frac{}{\vdash \forall x.A} \quad \frac{}{\vdash [E/x]A}$$

$$\frac{}{\vdash A} \quad \frac{}{\vdash B} \quad \frac{}{\vdash A \Rightarrow B} \quad \frac{}{\vdash B} \quad \frac{}{\vdash [E/x]A} \quad \frac{}{\vdash \exists x.A} \quad \frac{}{\vdash B} \quad \frac{}{\vdash B}$$

Automated Deduction - George Necula - Lecture 2 13

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 rule of consequence

$$\frac{\vdash A' \Rightarrow A \quad \vdash \{A\} c \{B\} \quad \vdash B \Rightarrow B'}{\vdash \{A'\} c \{B'\}}$$

Automated Deduction - George Necula - Lecture 2 14

Derivation Rules for Hoare Logic

- One rule for each syntactic construct:

$$\frac{}{\vdash \{A\} \text{ skip } \{A\}}$$

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

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

Automated Deduction - George Necula - Lecture 2 15

Derivation Rules for Hoare Logic (II)

- The rule for while is not syntax directed
 - It needs a loop invariant
- Exercise: try to see what is wrong if you make changes to the rule (e.g., drop " $\wedge b$ " in the premise, ...)

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{ while } b \text{ do } c \{A \wedge \neg b\}}$$

Automated Deduction - George Necula - Lecture 2 16

Hoare Rules: Assignment

- Example: $\{A\} x := x + 2 \{x \geq 5\}$. What is A ?
 - A has to imply $x \geq 3$
- General rule:

$$\frac{}{\vdash \{[e/x]A\} x := e \{A\}}$$

- Surprising how simple the rule is!
 - But try $\{A\} *x = 5 \{ *x + *y = 10 \}$
 - A is " $*y = 5$ or $x = y$ "
 - How come the rule does not work?
- Automated Deduction - George Necula - Lecture 2 17

Example: Assignment

- Assume that x does not appear in e
 - Prove that $\{\text{true}\} x := e \{x = e\}$
- We have

$$\frac{}{\vdash \{e = e\} x := e \{x = e\}}$$

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

- Assignment + consequence:

$$\frac{\vdash \text{true} \Rightarrow e = e \quad \vdash \{e = e\} x := e \{x = e\}}{\vdash \{\text{true}\} x := e \{x = e\}}$$

Automated Deduction - George Necula - Lecture 2 18

The Assignment Axiom (Cont.)

- Hoare said: "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."

- Caveats are sometimes needed for languages with aliasing:

- If x and y are aliased then
 - $\{ \text{true} \} x := 5 \{ x + y = 10 \}$
 - is true

Automated Deduction - George Necula - Lecture 2

19

Multiple Hoare Rules

- For some constructs multiple rules are possible:

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

(This was the "forward" axiom for assignment before Hoare)

$$\frac{}{\vdash A \wedge b \Rightarrow C} \quad \frac{}{\vdash \{C\} c \{A\}} \quad \frac{}{\vdash A \wedge \neg b \Rightarrow B}$$

$$\vdash \{A\} \text{while } b \text{ do } c \{B\}$$

$$\vdash \{C\} c \{b \Rightarrow C \wedge \neg b \Rightarrow B\}$$

$$\vdash \{b \Rightarrow C \wedge \neg b \Rightarrow B\} \text{while } b \text{ do } c \{B\}$$

- Exercise: these rules can be derived from the previous ones using the consequence rules

Automated Deduction - George Necula - Lecture 2

20

Example: Conditional

$$\frac{}{\vdash \{ \text{true} \wedge y \leq 0 \} x := 1 \{ x > 0 \}}$$

$$\frac{}{\vdash \{ \text{true} \wedge y > 0 \} x := y \{ x > 0 \}}$$

$$\vdash \{ \text{true} \} \text{if } y \leq 0 \text{ then } x := 1 \text{ else } x := y \{ x > 0 \}$$

- D_1 is obtained by consequence and assignment

$$\frac{}{\vdash \{1 > 0\} x := 1 \{x > 0\}} \quad \frac{}{\vdash \text{true} \wedge y \leq 0 \Rightarrow 1 > 0}$$

$$\vdash \{ \text{true} \wedge y \leq 0 \} x := 1 \{ x > 0 \}$$

- D_2 is also obtained by consequence and assignment

$$\frac{}{\vdash \{y > 0\} x := y \{x > 0\}} \quad \frac{}{\vdash \text{true} \wedge y > 0 \Rightarrow y > 0}$$

$$\vdash \{ \text{true} \wedge y > 0 \} x := y \{ x > 0 \}$$

Automated Deduction - George Necula - Lecture 2

21

Example: Loop

- We want to derive that

$$\vdash \{x \leq 0\} \text{while } x \leq 5 \text{ do } x := x + 1 \{x = 6\}$$

- Use the rule for while with invariant $x \leq 6$

$$\frac{}{\vdash x \leq 6 \wedge x \leq 5 \Rightarrow x + 1 \leq 6} \quad \frac{}{\vdash \{x + 1 \leq 6\} x := x + 1 \{x \leq 6\}}$$

$$\vdash \{x \leq 6 \wedge x \leq 5\} x := x + 1 \{x \leq 6\}$$

$$\vdash \{x \leq 6\} \text{while } x \leq 5 \text{ do } x := x + 1 \{x \leq 6 \wedge x > 5\}$$

- Then finish-off with consequence

$$\frac{}{\vdash x \leq 0 \Rightarrow x \leq 6} \quad \frac{}{\vdash x \leq 6 \wedge x > 5 \Rightarrow x = 6} \quad \frac{}{\vdash \{x \leq 6\} \text{while } \dots \{x \leq 6 \wedge x > 5\}}$$

$$\vdash \{x \leq 0\} \text{while } \dots \{x = 6\}$$

Automated Deduction - George Necula - Lecture 2

22

Another Example

- Verify that

$$\vdash \{A\} \text{while true do } c \{B\}$$

holds for any A , B and c

- We must construct a derivation tree

$$\frac{}{\vdash A \Rightarrow \text{true}} \quad \frac{}{\vdash \{ \text{true} \wedge \text{true} \} c \{ \text{true} \}}$$

$$\frac{}{\vdash \text{true} \wedge \text{false} \Rightarrow B} \quad \frac{}{\vdash \{ \text{true} \} \text{while true do } c \{ \text{true} \wedge \text{false} \}}$$

$$\vdash \{A\} \text{while true do } c \{B\}$$

- We need an additional lemma:

$$\forall c. \vdash \{ \text{true} \} c \{ \text{true} \}$$

- How do you prove this one?

Automated Deduction - George Necula - Lecture 2

23

GCD Example

- Let c be the program:

```
while (x ≠ y) do
  if (x ≤ y)
    then y := y - x
    else x := x - y
```

- We'll derive that

$$\vdash \{x = m \wedge y = n\} c \{x = \text{gcd}(m, n)\}$$

Automated Deduction - George Necula - Lecture 2

24

GCD Example (2)

- Crucial to select good loop invariant

- Let the precondition Pre be

$$x = m \wedge y = n$$

- Let the postcondition $Post$ be

$$x = \text{gcd}(m, n)$$

We use the loop invariant

$$I \stackrel{\text{def}}{=} \text{gcd}(x, y) = \text{gcd}(m, n)$$

GCD Example (3)

We first use the rule of consequence to obtain the subgoal

$$\vdash \{I\} c \{I \wedge \neg(x \neq y)\} \quad (1)$$

But we also need to prove

$$\vdash Pre \Rightarrow I \quad (2)$$

$$\vdash I \wedge \neg(x \neq y) \Rightarrow Post \quad (3)$$

Subgoal 2 reduces to

$$x = m \wedge y = n \Rightarrow \text{gcd}(x, y) = \text{gcd}(m, n)$$

Subgoal 3 reduces to

$$\text{gcd}(x, y) = \text{gcd}(m, n) \wedge x = y \Rightarrow x = \text{gcd}(m, n)$$

GCD Example (4)

Now we still have to derive subgoal 1:

$$\vdash \{I\} c \{I \wedge \neg(x \neq y)\}$$

We can apply the rule for `while` and we get the subgoal

$$\vdash \{I \wedge x \neq y\} d \{I\} \quad (4)$$

where d is the body of the loop:

```

if(x ≤ y)
  then y := y - x
  else x := x - y
  
```

GCD Example (5)

We can derive subgoal 4 using the rule for conditionals and we get two subgoals

$$\vdash \{I \wedge x \neq y \wedge x \leq y\} y := y - x \{I\} \quad (5)$$

$$\vdash \{I \wedge x \neq y \wedge x > y\} x := x - y \{I\} \quad (6)$$

Each of the subgoals 5 and 6 can be derived using the rule of consequence followed by assignment:

$$\vdash I \wedge x \neq y \wedge x \leq y \Rightarrow \text{gcd}(m, n) = \text{gcd}(x, y - x) \quad (7)$$

$$\vdash I \wedge x \neq y \wedge x > y \Rightarrow \text{gcd}(m, n) = \text{gcd}(x - y, y) \quad (8)$$

GCD Example (6)

$$\vdash I \wedge x \neq y \wedge x > y \Rightarrow \text{gcd}(m, n) = \text{gcd}(x - y, y)$$

- The above can be proved by realizing that

$$\text{gcd}(x, y) = \text{gcd}(x - y, y)$$

- Q.e.d.
- This completes the proof
- We used a lot of arithmetic
- We had to invent the loop invariants

- What about the proof for total correctness?

Hoare Rule for Function Call

- If no recursion we can inline the function call

$$\frac{f(x_1, \dots, x_n) = C_f \in \text{Program} \quad \{A\} C_f \{B[f/x]\}}{\vdash \{A[e_1/x_1, \dots, e_n/x_n]\} x := f(e_1, \dots, e_n) \{B\}}$$

- In general,
 1. each function f has a Pre_f and $Post_f$

$$\vdash \{Pre_f[e_1/x_1, \dots, e_n/x_n]\} x := f(e_1, \dots, e_n) \{Post_f[x/f]\}$$

2. For each function we check $\{Pre_f\} C_f \{Post_f\}$

Axiomatic Semantics in Presence of Side-Effects

Naïve Handling of Program State

- We allow memory read in assertions: $*x + *y = 5$
- We try:
 - $\{ A \} *x = 5 \{ *x + *y = 10 \}$
- A ought to be " $*y = 5$ or $x = y$ "
- The Hoare rule would give us:
 - $(*x + *y = 10) [5 / *x]$
 - $= 5 + *y = 10$
 - $= *y = 5$ (we lost one case)
- How come the rule does not work?

Handling Program State

- We cannot have side-effects in assertions
 - While creating the theorem we must remove side-effects!
 - But how to do that when lacking precise aliasing information?
- Important technique: Postpone alias analysis**
- Model the state of memory as a symbolic mapping from addresses to values:
 - If E denotes an address and M a memory state then:
 - $sel(M, E)$ denotes the contents of memory cell
 - $upd(M, E, V)$ denotes a new memory state obtained from M by writing V at address E

More on Memory

- We allow variables to range over memory states
 - So we can quantify over all possible memory states
- And we use the special pseudo-variable μ in assertions to refer to the current state of memory
- Example:
 - $\forall i. i \geq 0 \wedge i < 5 \Rightarrow sel(\mu, A + i) > 0$ = $allpositive(\mu, A, 0, 5)$
 - says that entries 0..4 in array A are positive

Semantics of Memory Expressions

- We need a new kind of values (memory values)

Values $v ::= n \mid a \mid \sigma$

$$\frac{}{\rho, \sigma \vdash \mu \Downarrow \sigma}$$

$$\frac{\rho, \sigma \vdash E_m \Downarrow \sigma' \quad \rho, \sigma \vdash E_a \Downarrow a}{\rho, \sigma \vdash sel(E_m, E_a) \Downarrow \sigma'(a)}$$

$$\frac{\rho, \sigma \vdash E_m \Downarrow \sigma' \quad \rho, \sigma \vdash E_a \Downarrow a \quad \rho, \sigma \vdash E_v \Downarrow v}{\rho, \sigma \vdash upd(E_m, E_a, E_v) \Downarrow \sigma'[a := v]}$$

Hoare Rules: Side-Effects

- To correctly model writes we use memory expressions
 - A memory write changes the value of memory

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

- Important technique: treat memory as a whole**
- And reason later about memory expressions with inference rules such as (McCarthy):

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

Memory Aliasing

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

$$\begin{aligned}
 A &= (*x + *y = 10)[\text{upd}(\mu, x, 5)/\mu] \\
 &= (\text{sel}(\mu, x) + \text{sel}(\mu, y) = 10) [\text{upd}(\mu, x, 5)/\mu] \\
 &= \text{sel}(\text{upd}(\mu, x, 5), x) + \text{sel}(\text{upd}(\mu, x, 5), y) = 10 \quad (*) \\
 &= 5 + \text{sel}(\text{upd}(\mu, x, 5), y) = 10 \\
 &= \text{if } x = y \text{ then } 5 + 5 = 10 \text{ else } 5 + \text{sel}(\mu, y) = 10 \\
 &= x = y \text{ or } *y = 5 \quad (**)
 \end{aligned}$$
- To (*) is theorem generation
- From (*) to (**) is theorem proving

Alternative Handling for Memory

- Reasoning about aliasing is expensive (NP-hard)
- Sometimes completeness is sacrificed with the following (approximate) rule:

$$\text{sel}(\text{upd}(M, E_1, E_2), E_3) = \begin{cases} E_2 & \text{if } E_1 = (\text{obviously}) E_3 \\ \text{sel}(M, E_3) & \text{if } E_1 \neq (\text{obviously}) E_3 \\ p & \text{otherwise (p is a fresh new parameter)} \end{cases}$$

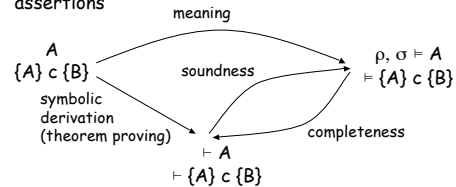
- The meaning of "obvious" varies:
 - The addresses of two distinct globals are \neq
 - The address of a global and one of a local are \neq
- PREFIX and GCC use such schemes

Using Hoare Rules. Notes

- Hoare rules are mostly syntax directed
- There are three wrinkles:
 - When to apply the rule of consequence?
 - What invariant to use for while?
 - How do you prove the implications involved in consequence?
- The last one 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.)

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



Soundness of Axiomatic Semantics

- Formal statement

$$\text{If } \vdash \{ A \} c \{ B \} \text{ then } \models \{ A \} c \{ B \}$$
- or, equivalently

$$\text{For all } \rho, \sigma, \text{ if } \rho, \sigma \models A \text{ and } D :: \rho, \sigma \vdash c \Downarrow \rho', \sigma' \text{ and } H :: \vdash \{ A \} c \{ B \} \text{ then } \rho', \sigma' \models B$$

Completeness of Axiomatic Semantics Weakest Preconditions

Completeness of Axiomatic Semantics

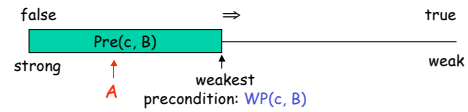
- Is it true that whenever $\models \{A\} c \{B\}$ we can also derive $\vdash \{A\} c \{B\}$?
- If it isn't then it means that there are valid properties of programs that we cannot verify with Hoare rules
- Good news: for our language the Hoare triples are complete
- Bad news: only if the underlying logic is complete (whenever $\models A$ we also have $\vdash A$)
 - this is called relative completeness

Automated Deduction - George Necula - Lecture 2

43

Proof Idea

- Dijkstra's idea: To verify that $\{A\} c \{B\}$
 - Find out all predicates A' such that $\models \{A'\} c \{B\}$
 - call this set $Pre(c, B)$
 - Verify for one $A' \in Pre(c, B)$ that $A \Rightarrow A'$
- Assertions can be ordered:



- Thus: compute $WP(c, B)$ and prove $A \Rightarrow WP(c, B)$

Automated Deduction - George Necula - Lecture 2

44

Proof Idea (Cont.)

- Completeness of axiomatic semantics:
 - If $\models \{A\} c \{B\}$ then $\vdash \{A\} c \{B\}$
- Assuming that we can compute $wp(c, B)$ with the following properties:
 1. wp is a precondition (according to the Hoare rules)
 - $\vdash \{wp(c, B)\} c \{B\}$
 2. wp is the weakest precondition
 - If $\models \{A\} c \{B\}$ then $\models A \Rightarrow wp(c, B)$
$$\frac{\vdash A \Rightarrow wp(c, B) \quad \vdash \{wp(c, B)\} c \{B\}}{\vdash \{A\} c \{B\}}$$
- We also need that whenever $\models A$ then $\vdash A$!

Automated Deduction - George Necula - Lecture 2

45

Weakest Preconditions

- Define $wp(c, B)$ inductively on c , following Hoare rules:

$$\frac{\{A\} c_1 \{C\} \quad \{C\} c_2 \{B\}}{\{A\} c_1; c_2 \{B\}}$$

$$wp(c_1; c_2, B) = wp(c_1, wp(c_2, B))$$

$$\frac{\{ [e/x]B \}}{\{ [e/x]B \} x := E \{B\}}$$

$$wp(x := e, B) = [e/x]B$$

$$\frac{\{A_1\} c_1 \{B\} \quad \{A_2\} c_2 \{B\}}{\{E \Rightarrow A_1 \wedge \neg E \Rightarrow A_2\} \text{ if } E \text{ then } c_1 \text{ else } c_2 \{B\}}$$

$$wp(\text{if } E \text{ then } c_1 \text{ else } c_2, B) = E \Rightarrow wp(c_1, B) \wedge \neg E \Rightarrow wp(c_2, B)$$

Automated Deduction - George Necula - Lecture 2

46

Weakest Preconditions for Loops

- We start from the equivalence
 - $\text{while } b \text{ do } c = \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}$
- Let $w = \text{while } b \text{ do } c$ and $W = wp(w, B)$
- We have that
 - $W = b \Rightarrow wp(c, W) \wedge \neg b \Rightarrow B$
- But this is a recursive equation !
 - We know how to solve these using domain theory
- We need a domain for assertions

Automated Deduction - George Necula - Lecture 2

47

A Partial-Order for Assertions

- What is the assertion that contains least information?
 - true - does not say anything about the state
- What is an appropriate information ordering?
 - $A \leq A' \text{ iff } \models A' \Rightarrow A$
- Is this partial order complete?
 - Take a chain $A_1 \leq A_2 \leq \dots$
 - Let $\bigwedge A_i$ be the infinite conjunction of A_i
 - $\sigma \models \bigwedge A_i \text{ iff for all } i \text{ we have that } \sigma \models A_i$
 - Verify that $\bigwedge A_i$ is the least upper bound
- Can $\bigwedge A_i$ be expressed in our language of assertions?
 - In many cases yes (see Winskel), we'll assume yes

Automated Deduction - George Necula - Lecture 2

48

Weakest Precondition for WHILE

- Use the fixed-point theorem
 - $F(A) = b \Rightarrow wp(c, A) \wedge \neg b \Rightarrow B$
 - Verify that F is both monotonic and continuous
- The least-fixed point (i.e. the weakest fixed point) is
 - $wp(w, B) = \wedge F^i(\text{true})$

Weakest Preconditions (Cont.)

- Define a family of wp's
 - $wp_k(\text{while } e \text{ do } c, B)$ = weakest precondition on which the loop if it terminates in k or fewer iterations, it terminates in B
 - $wp_0 = \neg E \Rightarrow B$
 - $wp_1 = E \Rightarrow wp(c, wp_0) \wedge \neg E \Rightarrow B$
 - ...
- $wp(\text{while } e \text{ do } c, B) = \wedge_{k \geq 0} wp_k = \text{lub} \{wp_k \mid k \geq 0\}$
- Is it the case that $wp_k \Rightarrow wp_{k-1}$? The opposite?
- Weakest preconditions are
 - Impossible to compute (in general)
 - Can we find something easier to compute yet sufficient?

Weakest Precondition. Example 1

- Consider the code:
 - while $x \leq 5$ do $x := x + 1$
 - with postcondition $P = x \geq 7$
- What is the weakest precondition?
- $wp(x := x + 1, A) = A[x+1/x]$
- $WP_0 = \neg(x \leq 5) \Rightarrow x \geq 7 = x \notin \{6\}$
- $WP_1 = x \leq 5 \Rightarrow x + 1 \notin \{6\} \wedge WP_0 = x \notin \{5, 6\}$
- $WP_2 = x \leq 5 \Rightarrow x + 1 \notin \{5, 6\} \wedge WP_0 = x \notin \{4, 5, 6\}$
- ...
- $WP = x \geq 7$

Weakest Precondition. Example 2

- Consider the code:
 - while $x \geq 5$ do $x := x + 1$
 - with postcondition $P = x \geq 7$
- What is the weakest precondition?
- $wp(x := x + 1, A) = A[x+1/x]$
- $WP_0 = \neg(x \geq 5) \Rightarrow x \geq 7 = x \geq 5$
- $WP_1 = x \geq 5 \Rightarrow x + 1 \geq 5 \wedge WP_0 = x \geq 5$
- ...
- $WP = x \geq 5$

Theorem Proving and Program Analysis (again)

- Predicates form a lattice:
 - $WP(s, B) = \text{lub}_*(\text{Pre}(s, B))$
- This is not obvious at all:
 - $\text{lub} \{P_1, P_2\} = P_1 \vee P_2$
 - $\text{lub } PS = \vee_{P \in PS} P$
 - But can we always write this with a finite number of \vee ?
- Even checking implication can be quite hard
- Compare with program analysis in which lattices are of finite height and quite simple

Program Verification is Program Analysis on the lattice of first order formulas