

Computer Science 203  
Programming Languages  
Fall 2005

Lecture 7  
Axiomatic Semantics

Cormac Flanagan

University of California, Santa Cruz

CMPS203 Lecture 7

1

### Review

---

- Operational semantics
  - relatively simple
  - many flavors
  - not compositional
- Good for reasoning about
  - properties of the language
    - eg. determinism
  - correctness of tools that manipulate programs
    - eg. compilers, type checkers, etc
- We would also like a semantics that is appropriate for reasoning about program correctness.

CMPS203 Lecture 7

2

### Axiomatic Semantics

---

- An axiomatic semantics consists of:
  - A language for making assertions about programs.
  - Rules for establishing the assertions.
- Some typical kinds of assertions:
  - This program terminates.
  - The variables  $x$  and  $y$  have the same value throughout the execution of the program whenever  $z$  is 0.
  - All array accesses are within array bounds.
- Some typical languages of assertions:
  - First-order logic.
  - Other logics (e.g., temporal logic).

CMPS203 Lecture 7

3

### History

---

- Program verification is almost as old as programming (e.g., "Checking a Large Routine", Turing 1949)
- In the late '60s, Floyd had rules for flowcharts and Hoare for a language similar to IMP.
- Since then, there have been axiomatic semantics for substantial languages, and many applications.

CMPS203 Lecture 7

4

### 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

CMPS203 Lecture 7

5

### Dijkstra Said

---

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

CMPS203 Lecture 7

6

## Hoare Also Said

- It has been found a serious problem to define these languages [ALGOL, FORTRAN, COBOL] with sufficient rigour to ensure compatibility among all implementors. Since the purpose of compatibility is to facilitate interchange of programs expressed in the language, one way to achieve this would be to insist that all implementations of the language shall "satisfy" the axioms and rules of inference which underlie proofs of the properties of programs expressed in the language, so that all predictions based on these proofs will be fulfilled, except in the event of hardware failure. In effect, this is equivalent to accepting the axioms and rules of inference as the ultimately definitive specification of the meaning of the language.
- Another of the objectives of formal language definition is to assist in the design of better programming languages.

## Other Applications of Axiomatic Semantics

- The project of defining and proving everything formally has not succeeded (at least not yet).
- Proving has not replaced testing (and praying).
- Some applications of axiomatic semantics:
  - Documentation of programs and interfaces.
  - Guidance in design and coding.
  - Proving the correctness of algorithms (or finding bugs).
  - Proving the correctness of hardware descriptions (or finding bugs).
  - "Extended static checking" (e.g., checking array bounds).
  - Proof-carrying code.

## Assertions for IMP

- Partial correctness assertion:**  $\{A\} c \{B\}$   
If A holds in state  $\sigma$   
and there exists  $\sigma'$  such that  $\langle c, \sigma \rangle \Downarrow \sigma'$   
then B holds in  $\sigma'$ .
- Total correctness assertion:**  $[A] c [B]$   
If A holds in state  $\sigma$   
then there exists  $\sigma'$  such that  $\langle c, \sigma \rangle \Downarrow \sigma'$   
and B holds in state  $\sigma'$ .
- 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

- We use first-order predicate logic on top of IMP arithmetic expressions

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

- All IMP boolean expressions are therefore assertions.

## Semantics of Assertions

- The judgement  $\sigma \models A$  means that an assertion holds in a 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 $\forall n \in \mathbb{Z}. \sigma[x:=n] \models A$
$\sigma \models \exists x.A$	iff $\exists n \in \mathbb{Z}. \sigma[x:=n] \models A$

## Semantics of Assertions

- Now we can define formally the meaning of a partial correctness assertion.

$$\begin{aligned}
 \models \{A\} c \{B\} \text{ holds if and only if} \\
 \forall \sigma \in \Sigma. \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]
 \end{aligned}$$

- ... and the meaning of a total correctness assertion.

$$\begin{aligned}
 \models [A] c [B] \text{ holds if and only if} \\
 \forall \sigma \in \Sigma. \sigma \models A \Rightarrow [\exists \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma' \wedge \sigma' \models B]
 \end{aligned}$$

- ... which simplifies to

$$\begin{aligned}
 \forall \sigma \in \Sigma. \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \sigma' \models B] \\
 \wedge \forall \sigma \in \Sigma. \sigma \models A \Rightarrow \exists \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma'
 \end{aligned}$$

## Deriving Assertions

- Formal definition of  $\models \{A\} c \{B\}$  is difficult to use
  - $\models \{A\} c \{B\}$  holds if and only if
    - $\forall \sigma \in \Sigma. \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma. \langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$
    - defined in terms of the operational semantics
    - $\neg \forall \sigma$
- So we define a symbolic technique (ie, a **logic**) for deriving valid triples  $\vdash \{A\} c \{B\}$  from other valid triples.

## 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**:

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

## Derivation Rules for Hoare Logic $\vdash \{A\} c \{B\}$

- One rule for each language construct (plus one more)

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

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

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

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

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

Rule of Consequence

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

## Free and Bound Variables

- One of the key notions in logic and in programming languages is variable scoping and substitution.
- Assertions are equivalent up to renaming of bound variables (called alpha-renaming)
- Examples:
  - $\forall x. x = x$  is the same as  $\forall y. y = y$
  - $\forall x. \forall y. x = y$  is the same as  $\forall z. \forall x. z = x$

## Substitution

- $[e'/x]e$  is the result of substituting  $e'$  for  $x$  in  $e$ .
  - It is also written, for example,  $e[e'/x]$  and  $e(e'/x)$ .
  - We substitute only the free occurrences!
- We may need to alpha-rename bound variables in order to avoid conflicts.
  - Before substituting  $[e'/x]$  in  $\forall y. x = y$  we may rename  $y$  in case it occurs in  $e'$ .
  - The result is  $\forall z. e' = z$ .
- We say that substitution avoids variable capture
  - $[x/z] \forall x. z = x$  is  $\forall x. x = x$ . **Bad!**
  - $[x/z] \forall x. z = x$  is  $\forall y. x = y$ . **Good!**

## 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.)

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

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

### Example: Assignment

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

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

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

- Assignment + consequence:

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

### 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 needed for languages with aliasing:
  - If  $x$  and  $y$  are aliased then
    - $\{\text{true}\} x := 5 \{x + y = 10\}$  is true

### Example: Conditional

$$\frac{D_1 :: \vdash \{\text{true} \wedge y \leq 0\} x := 1 \{x > 0\} \quad D_2 :: \vdash \{\text{true} \wedge y > 0\} x := y \{x > 0\}}{\vdash \{\text{true}\} \text{if } y \leq 0 \text{ then } x := 1 \text{ else } x := y \{x > 0\}}$$

- $D_1$  is obtained by consequence and assignment

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

- $D_2$  is also obtained by consequence and assignment

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

### Example: Loop

- We want to derive that
  - $\vdash \{x \leq 0\} \text{while } x \leq 5 \text{ do } x := x + 1 \{x = 6\}$
- We use the rule for while with invariant  $x \leq 6$ :

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

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

- We finish off with consequence:

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

### 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

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

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

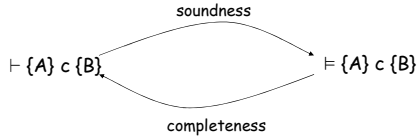
- We need an additional lemma:
  - $\forall A. \forall c. \vdash \{A\} c \{\text{true}\}$

### Notes on Using Hoare Rules

- 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 involves theorem proving:
  - This turns out to be doable.
  - The loop invariants turn out to be the hardest problem! (Should the programmer give them?)

## Where Do We Stand?

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



## Soundness of Axiomatic Semantics

- Formal statement of soundness:  
If  $\vdash \{A\} c \{B\}$  then  $\models \{A\} c \{B\}$ .
- or, equivalently  
For all  $\sigma$ , if  $\sigma \models A$  and  $D :: \langle c, \sigma \rangle \Downarrow \sigma'$   
and  $H :: \vdash \{A\} c \{B\}$  then  $\sigma' \models B$ .
- Proof: simultaneous induction on the structure of  $D$  and  $H$ .

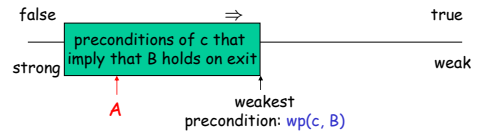
## Completeness of Axiomatic Semantics

- Formal statement of completeness:  
If  $\models \{A\} c \{B\}$  then  $\vdash \{A\} c \{B\}$ .
- or, equivalently  
Suppose that, for all  $\sigma, D, \sigma'$ ,  
if  $\sigma \models A$  and  $D :: \langle c, \sigma \rangle \Downarrow \sigma'$  then  $\sigma' \models B$ .  
Then there exists  $H$  such that  $H :: \vdash \{A\} c \{B\}$ .
- Proof: harder, and requires an assumption that says that loop invariants can be expressed as logical formulas.

(See slides on-line and Winskel's book for more.)

## Weakest Preconditions (Dijkstra)

- Assertions can be ordered:



- Thus: to verify  $\{A\} c \{B\}$ , we may compute  $wp(c, B)$  and prove  $A \Rightarrow wp(c, B)$ .

## Weakest Preconditions

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

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

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

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

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

$$\frac{\{A\} c_1 \{B\} \quad \{A'\} c_2 \{B\}}{\{E \Rightarrow A \wedge \neg E \Rightarrow A'\} \text{ if } E \text{ then } s_1 \text{ else } s_2 \{B\}}$$

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

## 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  $W = wp(\text{while } b \text{ do } c, B)$
- 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... in a complete partial order.
  - Least solutions may or may not exist over formulas.

## Reading and Homework

- "Guarded Commands, Nondeterminacy and Formal Derivation of Programs"
  - Dijkstra
- "Extended Static Checking for Java"
  - an application of some of the ideas described so far
- Project Proposal

## Homework 5 (for October 25)

1. Using Hoare's rules, prove  
 $\{x = y\}$   
 $x := x + 1; y := y + 1;$   
 $\{x = y\}$
2. Using Hoare's rules, prove  
 $\{y = z\}$  while b do  $y := y - x \{ \exists k. z = (y + (k * x)) \}$   
for an arbitrary b.

## Extra material on axiomatic semantics: soundness and completeness

(covered in the lectures  
only as time permits)

## Hoare Rules: Assignment and References

- When is the following Hoare triple valid?  
 $\{A\} *x = 5 \{ *x + *y = 10 \}$
- A ought to be  $*y = 5$  or  $x = y$
- The Hoare rule for assignment would give us:  
 $[5/*x](*x + *y = 10)$   
 $= 5 + *y = 10$   
 $= *y = 5$  (we lost one case)
- How come the rule does not work?

## Hoare Rules: Assignment and References (Cont.)

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

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

- Important technique:
  - Treat memory as a whole.
  - And reason about memory expressions with rules such as McCarthy's:

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

## Memory Aliasing

- Consider again:  $\{A\} *x := 5 \{ *x + *y = 10 \}$
- We obtain:  
 $A = [\text{upd}(M, x, 5)/M] (*x + *y = 10)$   
 $= [\text{upd}(M, x, 5)/M] (\text{sel}(M, x) + \text{sel}(M, y) = 10)$   
 $= \text{sel}(\text{upd}(M, x, 5), x) + \text{sel}(\text{upd}(M, x, 5), y) = 10$   
 $= 5 + \text{sel}(\text{upd}(M, x, 5), y) = 10$   
 $= \text{if } x = y \text{ then } 5 + 5 = 10 \text{ else } 5 + \text{sel}(M, y) = 10$   
 $= x = y \text{ or } *y = 5$

## Mutable Records

- Let  $r : \text{RECORD } f1 : T1; f2 : T2 \text{ END}$
- One method for handling records:
  - One "memory" for each field
  - The record address is the index
  - $r.f1$  is  $\text{sel}(f1,r)$  and  $r.f1 := E$  is  $f1 := \text{upd}(f1,r,E)$

CMPS201 Lecture 7

37

## Soundness of Axiomatic Semantics

- Formal statement  
If  $\vdash \{ A \} c \{ B \}$  then  $\models \{ A \} c \{ B \}$ .
- or, equivalently  
For all  $\sigma$ , if  $\sigma \models A$  and  $D :: \langle c, \sigma \rangle \Downarrow \sigma'$   
and  $H :: \vdash \{ A \} c \{ B \}$  then  $\sigma' \models B$ .
- How can we prove this?
  - By induction on the structure of  $c$ ?
    - No, problems with while and rule of consequence.
  - By induction on the structure of  $D$ ?
    - No, problems with rule of consequence.
  - By induction on the structure of  $H$ ?
    - Not quite, problems with while.
  - By simultaneous induction on the structure of  $D$  and  $H$ .

CMPS201 Lecture 7

38

## Simultaneous Induction

- Consider two structures  $D$  and  $H$ .
  - Assume that  $x < y$  iff  $x$  is a substructure of  $y$ .
- Define the lexicographic ordering  
 $(d, h) < (d', h')$  iff  $d < d'$  or  $d = d'$  and  $h < h'$
- This is a well-founded order and serves for a simultaneous induction.
- If  $d < d'$  then  $h$  can actually be larger than  $h'$ .
- It can even be unrelated to  $h'$ .

CMPS201 Lecture 7

39

## Soundness of the Consequence Rule

- Case: last rule used in  $H :: \vdash \{ A \} c \{ B \}$  is the consequence rule:
 
$$\frac{\vdash A \Rightarrow A' \quad H_1 :: \vdash \{ A' \} c \{ B' \} \quad \vdash B' \Rightarrow B}{\vdash \{ A \} c \{ B \}}$$
- From soundness of the first-order logic derivations we have  $\sigma \models A \Rightarrow A'$ , hence  $\sigma \models A'$
- From IH with  $H_1$  and  $D$  we get that  $\sigma' \models B'$ .
- From soundness of the first-order logic derivations we have that  $\sigma' \models B' \Rightarrow B$ , hence  $\sigma' \models B$ , q.e.d.

CMPS201 Lecture 7

40

## Soundness of the Assignment Axiom

- Case: the last rule used in  $H :: \vdash \{ A \} c \{ B \}$  is the assignment rule

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

- The last rule used in  $D :: \langle x := e, \sigma \rangle \Downarrow \sigma'$  must be

$$\frac{D_1 :: \langle e, \sigma \rangle \Downarrow n}{\langle x := e, \sigma \rangle \Downarrow \sigma[x := n]}$$

- We must prove the **substitution lemma**:  
If  $\sigma \models [e/x]B$  and  $\langle \sigma, e \rangle \Downarrow n$  then  $\sigma \models [n/x]B$

CMPS201 Lecture 7

41

## Soundness of the While Rule

- Case: last rule used in  $H :: \vdash \{ A \} c \{ B \}$  is the while rule:

$$\frac{H_1 :: \vdash \{ A \wedge b \} c \{ A \}}{\vdash \{ A \} \text{while } b \text{ do } c \{ A \wedge \neg b \}}$$

- There are two possible rules at the root of  $D$ .
  - We do only the complicated case:

$$\frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{true} \quad D_2 :: \langle c, \sigma \rangle \Downarrow \sigma' \quad D_3 :: \langle \text{while } b \text{ do } c, \sigma' \rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''}$$

CMPS201 Lecture 7

42

### Soundness of the While Rule (Cont.)

Assume that  $\sigma \models A$

To show that  $\sigma'' \models A \wedge \neg b$

- By property of booleans and  $D_1$  we get  $\sigma \models b$ .
    - Hence  $\sigma \models A \wedge b$ .
  - By IH on  $H_1$  and  $D_2$  we get  $\sigma' \models A$ .
  - By IH on  $H$  and  $D_3$  we get  $\sigma'' \models A \wedge \neg b$ , q.e.d.
- Note that in the last use of IH the derivation  $H$  did not decrease.
- See Winskel, Chapter 6.5 for a soundness proof with denotational semantics.

### 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.
  - The underlying logic must also be expressive enough.

### A Partial Order for Assertions

- What is the assertion that contains least information?
  - true - it does not say anything about the state.
- What is an appropriate information ordering?
  - $A \sqsubseteq A'$  iff  $\models A' \Rightarrow A$
- Is this partial order complete?
  - Take a chain  $A_1 \sqsubseteq A_2 \sqsubseteq \dots$
  - Let  $\bigwedge A_i$  be the infinite conjunction of  $A_i$ 
    - $\sigma \models \bigwedge A_i$  iff for all  $i$  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 so.

### 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 monotone and continuous.
- The least fixed point (i.e., the weakest fixed point) is
  - $wp(\text{while } b \text{ do } c, B) = \bigwedge F^i(\text{true})$
- Notice that unlike for the denotational semantics of IMP, we are not working on a flat domain.

### Proof Idea for Completeness

- 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  $\vdash A$ .