Introduction to Axiomatic Semantics

Meeting 10, CSCI 5535, Spring 2009

Announcements

- Homework 3 due tonight
- Homework 2 is graded
  - 13 (mean), 14 (median), out of 21 total, but...
  - Graduate class: final project is what counts, homeworks are to check understanding
  - Most difficult: comparing big-step and contextual for IMP with exceptions
    - Many did not answer the question, rather echoed general comparisons made in class.
    - “Both your ideas and the clarity with which they are expressed matter.” (like research papers)

Questions?

Plan for Axiomatic Semantics

- History and Motivation (last time)
- Assertions (today)
- Validity (today)
- Derivation Rules (today)
- Soundness (next time)
- Completeness (next time)

Axiomatic Semantics

A semantics that is appropriate for arguing program correctness

One-Slide Summary

- An axiomatic semantics consists of:
  - A language for stating assertions about programs,
  - Rules for establishing the truth of assertions
- Some typical kinds of assertions:
  - This program terminates
  - If this program terminates, the variables x and y have the same value throughout the execution of the program
  - The array accesses are within the array bounds
- Some typical languages of assertions
  - First-order logic
  - Other logics (temporal, linear, pointer-assertion)
  - Special-purpose specification languages (SLIC, Z, Larch)
Another Tony Hoare Quote

"It has been found a serious problem to define these languages [ALGOL, FORTRAN, COBOL] with sufficient rigor to ensure compatibility among all implementations. ... 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 properties of programs expressed in the language. In effect, this is equivalent to accepting the axioms and rules of inference as the ultimately definitive specification of the meaning of the language."

Do you believe this?

- Would such language specifications be useful to you?
  - Language should be different
  - Provide basis for checking
  - Hard to use - for higher-level?"

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 debugging
- Applications of axiomatic semantics:
  - 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
  - Documentation of programs and interfaces

Notation: Assertions

\{A\} c \{B\}

with the meaning that:
- if A holds in state \(\sigma\) and if \(<c, \sigma> \Downarrow \sigma'\)
  - 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

Assertions for IMP

- \{A\} c \{B\} is a partial correctness assertion.
  - Doesn’t imply termination (it is valid if \(c\) diverges)
- \[A\] c \{B\} is a total correctness assertion meaning that
  - If A holds in state \(\sigma\)
    - Then there exists \(\sigma'\) such that \(<c, \sigma> \Downarrow \sigma'\) and B holds in state \(\sigma'\)
- Now let us be more formal (you know you want it!)
  - Formalize the language of assertions, A and B
  - Say when an assertion holds in a state
  - Give rules for deriving Hoare triples
The Assertion Language

- We use first-order predicate logic on top of IMP expressions
  \[ A ::= \text{true} \mid \text{false} \mid e_1 = e_2 \mid e_1 \geq e_2 \mid A_1 \land A_2 \mid A_1 \lor A_2 \mid A_1 \Rightarrow A_2 \mid \forall x.A \mid \exists x.A \]

- Note that we are somewhat sloppy in mixing logical variables and the program variables

- All IMP variables implicitly range over integers
- All IMP boolean expressions are also assertions

Semantics of Assertions: \( \models \)

- Need to assign meanings to our assertions
- Relation \( \sigma \models A \) to say that an assertion holds in a given state (= “\( A \) is true in \( \sigma \)”) – This is well-defined when \( \sigma \) is defined on all variables occurring in \( A \)

- The \( \models \) relation is defined inductively on the structure of assertions (surprise!)
- It relies on the denotational semantics of arithmetic expressions from IMP

Formal Definition of \( \models \)

- \( \sigma \models \text{true} \) always
- \( \sigma \models e_1 = e_2 \) iff \( \sigma[e_1] = \sigma[e_2] \)
- \( \sigma \models e_1 \geq e_2 \) iff \( \sigma[e_1] \geq \sigma[e_2] \)
- \( \sigma \models A_1 \land A_2 \) iff \( \sigma \models A_1 \) and \( \sigma \models A_2 \)
- \( \sigma \models A_1 \lor A_2 \) 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 \( \exists n \in \mathbb{Z}. \sigma[x:=n] \models A \)

Hoare Triple Semantics

- Now we can define formally the meaning of a partial correctness assertion \( \models \{ A \} c \{ B \} \)
  \( \forall \sigma \in \Sigma. \forall a' \in \Sigma. (\sigma \models A \land \langle c, \sigma \rangle \uparrow \sigma') \Rightarrow \sigma' \models B \)
  - and a total correctness assertion \( \models \{ A \} c \{ B \} \)
  \( \forall \sigma \in \Sigma. \sigma \models A \Rightarrow \exists a' \in \Sigma. \langle c, \sigma \rangle \uparrow \sigma' \land \sigma' \models B \)

- or even better yet: (explain this to me!)
  \( \forall \sigma \in \Sigma. \forall a' \in \Sigma. (\sigma \models A \land \langle c, \sigma \rangle \uparrow \sigma') \Rightarrow \sigma' \models B \land \forall \sigma \in \Sigma. \sigma \models A \Rightarrow \exists a' \in \Sigma. \langle c, \sigma \rangle \uparrow \sigma' \)

Deriving Assertions

- Have a formal mechanism to decide \( \models \{ A \} c \{ B \} \)
  - But it is not satisfactory. Why?
  - Guarantee termination?
  - For total correctness?
  - How to construct \( \sigma \)
  - Use evaluation

-
Deriving Assertions

- Have a formal mechanism to decide $\vdash \{A\} \triangleright \{B\}$
  - But it is not satisfactory
  - Because $\vdash \{A\} \triangleright \{B\}$ is defined in terms of the operational semantics, we practically have to run the program to verify an assertion
  - It is impossible to effectively verify the truth of a $\forall x. A$ assertion (check every integer?)
- Plan: define a symbolic technique for deriving valid assertions from others that are known to be valid
  - We start with validity of first-order formulas

Derivation Rules

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

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

First-Order Rules

- Similarly we write $\vdash \{A\} \triangleright \{B\}$ 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

Propositional Rules

- $\vdash A \rightarrow B$
  - $\vdash A \land B$
  - $\vdash A$
  - $\vdash A 
  - $\vdash A 
  - $\vdash B$
  - $\vdash A 
  - $\vdash B$
  - $\vdash A 

Natural deduction style

- $\vdash A \rightarrow B$
Derivation Rules for Hoare Logic

• One rule for each syntactic construct:

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

Alternate Hoare Rules

• For some constructs multiple rules are possible:

\[ \vdash \{ A \} x := e \{ \exists x_0. (x_0/x)A \land x = [x_0/x]e \} \]
(This one is called the “forward” axiom for assignment)

\[ \vdash \{ A \} \text{while } b \text{ do } c \{ A \land \neg b \Rightarrow B \} \]
(C is the loop invariant)

Example Verifications

Example: Assignment

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

\[ \vdash \{ \text{true} \} x := e \{ x = e \} \]
Example: Assignment

• (Assuming that x does not appear in e)
  Prove that \( \{ \text{true} \} \ x := e \ \{ x = e \} \)
• Assignment Rule:
  \[ \vdash (e = e) \ x := e \ (x = e) \]
  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
    \( \{ \text{true} \} \ x := 5 \ (x + y = 10) \)
    is true

For Next Time

• Homework 4 out tonight due Mon Feb 23
• Think about possible projects