Introduction to Axiomatic Semantics
Meeting 15, CSCI 5535, Spring 2010

Announcements
• Homework 5 is due Thu
• Want to learn more about domain theory and fixed points?
  • Robert is speaking in the Type class today
  • after class (3:20) to 5:00 in ECEE 265
• Time Spent:
  - HW2: 8.6 hrs mean, 3.1 stddev, 8.5 median
  - HW3: 9.1 hrs mean, 3.6 stddev, 9 median
  - HW4: 11.1 hrs mean, 6 stddev, 10 median

Questions?

Plan for Axiomatic Semantics
• History and Motivation
• Assertions
• Validity
• Derivation Rules
• Soundness
• Completeness

Review by Class Participation
• Tell Me About Operational Semantics
  "Step 1. Begin
  "Step 2. Stop
  "Step 3. Go"
• Tell Me About Structural Induction
  An inductive proof technique
  Induction over commands or derivations
  (syntactic entities)
• Tell Me About Denotational Semantics
  A language model
  Functions G and C
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)

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 flow-charts and Hoare for structured languages
- Since then, there have been axiomatic semantics for substantial languages, and many applications
  - ESC/Java, SLAM, PCC, SPARK Ada, ...

Tony Hoare Quote

"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

Agree or Disagree?

Edsger Dijkstra Quote

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

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\} \implies \{B\}

with the meaning that:
- if A holds in state \(\sigma\) and if \(<c, \sigma> \implies \sigma'\)
- 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\} \implies \{B\} is a **partial correctness assertion**
  - Doesn’t imply termination (\(c\) is valid if \(c\) diverges)
- \[A\] \implies [B] is a **total correctness assertion**
  meaning that
  - If A holds in state \(\sigma\)
  - Then there exists \(\sigma'\) such that \(<c, \sigma> \implies \sigma'\)
  - 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
  \[A ::= true | false | e_1 = e_2 | e_1 \geq e_2 \]
  \[| A_1 \land A_2 | A_1 \lor A_2 | A_1 \Rightarrow A_2 | \forall x.A | \exists x.A\]
- Note that we are somewhat sloppy in mixing logical variables and the program variables
  - Winskel introduces logical variables
- All IMP variables implicitly range over integers
- All IMP boolean expressions are also assertions

Semantics of Assertions:

- Need to assign meanings to our assertions
- Relation \(\models A\) 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
Deriving Assertions

- Have a formal mechanism to decide $\vdash \{A\} \subset \{B\}$
- But it is not satisfactory.
- Because $\vdash \{A\} \subset \{B\}$ 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
About your classmates

- Proficient in 5 natural languages
- Almost became a chef
- Grew up in Kuwait
- Has a pet cockatiel named Phoenix
- Is a big history enthusiast
- Plays guitar and bass
- Owns online business specializing in tacky holiday sweaters

Derivation Rules

- We write $\vdash A$ when $A$ can be derived from basic axioms ($\vdash A$ === “we can prove $A$”)

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

First-Order Rules

\[ \vdash [a/x]A \quad (a \text{ is fresh}) \]

\[ \vdash \forall x.A \quad \forall I \]

\[ \vdash [e/x]A \]

\[ \vdash \exists x.A \quad \exists E \]