Axiomatic Semantics: Verification Conditions
Meeting 18, CSCI 5535, Spring 2010

Announcements
• Homework 6 is due tonight
• Today’s forum: papers on automated testing using symbolic execution
• Anyone looking for a group?
• HW5
  – Time: 12.2 hrs mean, 4.9 stddev, 12 median

Questions?

Review of Soundness of Axiomatic Semantics

One-Slide Summary
• A system of axiomatic semantics is sound if everything we can prove is also true.
  \[(A \implies (B \implies (\phi \implies (\psi \implies \theta)) \implies \psi))\]
• We prove this by nested induction on the structure of the operational semantics derivation and the axiomatic semantics proof.
• A system of axiomatic semantics is complete if we can prove all true things.
  \[(A \implies (B \implies (\phi \implies (\psi \implies \theta)) \implies \psi))\]
• Our system is relatively complete (as just as complete as the underlying logic). We use weakest preconditions to reason about soundness. Verification conditions are preconditions that are easy to compute.

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

\[\begin{align*}
\text{soundness} & \iff (A \implies (B \implies (\phi \implies (\psi \implies \theta)) \implies \psi)) \\
\text{completeness} & \iff (A \implies (B \implies (\phi \implies (\psi \implies \theta)) \implies \psi)) \\
\end{align*}\]
End of Review

Proof Idea

• Dijkstra’s idea: To verify that \(( A ) \implies ( B )\)
a) Find out all predicates \( A’\) such that \(( A’ ) \implies ( B )\)
   - call this set \( \text{Pre}(c, B) \)
b) Verify for one \( A’\in \text{Pre}(c, B) \) that \( A \implies A’\)
• Assertions can be ordered:
  - true
  - false
  - weakest precondition \( \text{wp}(c, B) \)
  - strongest
• Thus: compute \( \text{wp}(c, B) \) and prove \( A \implies \text{wp}(c, B) \)

Completeness of Axiomatic Semantics

• If \( \models ( A ) \implies ( B ) \) can we always derive \( \vdash ( A ) \implies ( B ) \)?
• If so, axiomatic semantics is \textbf{complete}
• If not then 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 we can decide the underlying logic
     (given an oracle to decide \( \models A \))
     - this is called \textbf{relative completeness}

Weakest Preconditions

• Define \( \text{wp}(c, B) \) inductively on \( c \), following the Hoare rules:
  - \( \text{wp}(c_1 ; c_2 , B) = (A_1 ) c_1 (C) (A_2 ) c_2 (B) \)
  - \( \text{wp}(x := e , B) = \{ [e/x]B \} x := E (B) \)
  - \( \text{wp}(\text{if } b \text{ then } c_1 \text{ else } c_2 , B) = \{ b \implies c_1 (B) \} \land \neg b \implies \text{wp}(c_2 , B) \)
• Define \( \text{wp}(c, B) \) inductively on \( c \), following the Hoare rules:
  - \( \text{wp}(c_1 ; c_2 , B) = (A_1 ) c_1 (C) (A_2 ) c_2 (B) \)
  - \( \text{wp}(x := e , B) = \{ [e/x]B \} x := E (B) \)
  - \( \text{wp}(\text{if } E \text{ then } c_1 \text{ else } c_2 , B) = \{ E \implies c_1 (B) \land \neg E \implies c_2 (B) \}

Proof Idea

• \textbf{Completeness} of axiomatic semantics:
  \[
  \text{If } \models ( A ) \implies ( B ) \text{ then } \vdash ( A ) \implies ( B )
  \]
• Assuming that we can compute \( \text{wp}(c, B) \) with the following properties:
  - \( \text{wp} \) is a precondition (according to the Hoare rules)
    \( \{ \text{wp}(c, B) \} c ( B ) \)
  - \( \text{wp} \) is \textbf{(truly) the weakest} precondition
    \( \text{If } \models ( A ) \implies ( B ) \text{ then } \models A \implies \text{wp}(c, B) \)
    \[
    \vdash ( A ) \implies \text{wp}(c, B) \implies (\text{wp}(c, B) \implies c (B))
    \]
    \[
    \vdash ( A ) \implies ( B )
    \]

Weakest Preconditions

• \( \text{wp}(c_1 ; c_2 , B) = (A_1 ) c_1 (C) (A_2 ) c_2 (B) \)
• \( \text{wp}(x := e , B) = \{ [e/x]B \} x := E (B) \)
• \( \text{wp}(\text{if } E \text{ then } c_1 \text{ else } c_2 , B) = \{ E \implies c_1 (B) \land \neg E \implies c_2 (B) \}

Weakest Preconditions

• Define \( \text{wp}(c, B) \) inductively on \( c \), following the Hoare rules:
  - \( \text{wp}(c_1 ; c_2 , B) = (A_1 ) c_1 (C) (A_2 ) c_2 (B) \)
  - \( \text{wp}(x := e , B) = \{ [e/x]B \} x := E (B) \)
  - \( \text{wp}(\text{if } E \text{ then } c_1 \text{ else } c_2 , B) = \{ E \implies c_1 (B) \land \neg E \implies c_2 (B) \}

Weakest Preconditions for while

- We start from the unwinding 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 \( W = \text{wp}(w, B) \)
- We have that
  \[ W = b \land \text{wp}(c, W) \land \neg b \Rightarrow B \]
- But this is a recursive equation!
  - We know how to solve these using domain theory
  - But we need a domain for assertions

Weakest Precondition for while

- Use the fixed-point theorem
  \[ F(A) = b \Rightarrow \text{wp}(c, A) \land \neg b \Rightarrow B \]
  - (Where did this come from? Three slides back!)
  - I assert that \( F \) is both monotonic and continuous
- The least-fixed point (= the weakest fixed point) is
  \[ \text{wp}(w, B) = \bigwedge F(\text{true}) \]
  - Bottom in our ordering

A Partial-Order for Assertions

- Which assertion contains the least information?
  - “true” – does not say anything about the state
- What is an appropriate information ordering?
  - \( A \sqsubseteq A' \iff A \Rightarrow A' \)
- Is this partial order complete?
  - Take a chain \( A_1 \sqsubseteq A_2 \sqsubseteq \ldots \)
  - Let \( \bigwedge A_i \) be the infinite conjunction of \( A_i \)
  - I assert that \( \bigwedge A_i \) is the least upper bound
- Can \( \bigwedge A_i \) be expressed in our language of assertions?
  - Often yes (see Winskel), we’ll assume yes for now

Weakest Preconditions

- Define a family of wp’s
  \[ \text{wp}(\text{while } e \text{ do } c, B) = \text{weakest precondition on which the loop terminates in } B \text{ if it terminates in } k \text{ or fewer iterations} \]
  \[ \text{wp}_0 = E \Rightarrow B \]
  \[ \text{wp}_1 = E \Rightarrow \text{wp}(c, \text{wp}_0) \land \neg E \Rightarrow B \]
  - \( \ldots \)
  \[ \text{wp}(\text{while } e \text{ do } c, B) = \bigwedge_{k \geq 0} \text{wp}_k = \text{lub} \{\text{wp}_k \mid k \geq 0\} \]
- See notes on the web page for the proof of completeness with weakest preconditions
- Weakest preconditions are
  - Impossible to compute (in general)
  - Can we find something easier to compute yet sufficient?

About your classmates

- At one point nationally ranked in tennis
- Rated 2 dan from American Go Association
- Listens almost exclusively to early music
  (Medieval, Renaissance, Baroque)
- Has a 5 year old who builds amazing things with Legos
- Grew up in Santa Claus, Indiana
- Spent 2 weeks in Mexico studying how professors teach English
- BS from Christopher Newport Univ

Verification Condition Generation
Not Quite Weakest Preconditions

• Recall what we are trying to do:
  \[ \text{false} \rightarrow \text{true} \]

  weakest precondition: \( \text{WP}(c, B) \)

  verification condition: \( \text{VC}(c, B) \)

• We shall construct a verification condition: \( \text{VC}(c, B) \)
  - The loops are annotated with loop invariants
  - \( \text{VC} \) is guaranteed stronger than \( \text{WP} \)
  - But hopefully still weaker than \( A \):
    \[ A \rightarrow \text{VC}(c, B) \rightarrow \text{WP}(c, B) \]

Groundwork

• Factor out the hard work
  - Loop invariants
  - Function specifications (pre- and post-conditions)
• Assume programs are annotated with such specs
  - Good software engineering practice anyway
  - Requiring annotations = Kiss of Death?
• New form of while that includes a loop invariant:
  \[
  \text{while}_{\text{Inv}} \ b \ do \ c
  \]
  - Invariant formula \( \text{Inv} \) must hold every time before \( b \) is evaluated
  - A process for computing \( \text{VC}(\text{annotated}_c, \text{post}_c) \) is called \( \text{VCGen} \)

Verification Condition Generation

• Mostly follows the definition of the \( \text{wp} \) function:
  \[
  \begin{align*}
  \text{VC}(\text{skip}, B) &= B \\
  \text{VC}(c_1; c_2, B) &= \text{VC}(c_1, \text{VC}(c_2, B)) \\
  \text{VC}(\text{if } b \text{ then } c_1 \text{ else } c_2, B) &= b \Rightarrow \text{VC}(c_1, B) \land \neg b \Rightarrow \text{VC}(c_2, B) \\
  \text{VC}(x := e, B) &= [e/x]B \\
  \text{VC}(\text{while}_{\text{Inv}} b \ do \ c, B) &= ?
  \end{align*}
  \]

Example of VCGen

• By the sequencing rule, first we do the while loop (call it \( w \)):
  \[
  \begin{align*}
  \text{while}_{x+y=2} y > 0 \ do \\
  & y := y - 1; \\
  & x := x + 1
  \end{align*}
  \]
  \[
  \begin{align*}
  \text{VCGen}(w, x = 0) &= x+y=2 \land \\
  \forall x, y. \ x+y=2 &\Rightarrow (y=0 \Rightarrow \text{VC}(c, x+y=2)) \land y \leq 0 \Rightarrow x = 0) \\
  \text{VCGen}(y := y-1; x := x+1, x+y=2) &= (x+1) + (y-1) = 2 \\
  \text{w Result: } x+y=2 \land &\\
  \forall x, y. x+y=2 &\Rightarrow (y=0 \Rightarrow (x+1)+(y-1)=2) \land y \leq 0 \Rightarrow x = 0)
  \end{align*}
  \]

Example VCGen Problem

• Let’s compute the VC of this program with respect to post-condition \( x \neq 0 \)

  \[
  \begin{align*}
  x &:= 0; \\
  y &:= 2; \\
  \text{while}_{x+y=2} y > 0 \ do \\
  & y := y - 1; \\
  & x := x + 1
  \end{align*}
  \]
  First, what do we expect? What pre-condition do we need to ensure \( x \neq 0 \) after this?
Example of VCGen

• VC(w, x \neq 0) = x+y=2 \land
  \forall x, y. x+y=2 \Rightarrow
  (y>0 \Rightarrow (x+1)+(y-1)=2 \land y\leq0 \Rightarrow x=0)
• VC(x := 0; y := 2 ; w, x \neq 0) =
  0+2=2 \land
  \forall x, y. x+y=2 \Rightarrow
  (y>0 \Rightarrow (x+1)+(y-1)=2 \land y\leq0 \Rightarrow x=0)
• So now we ask an automated theorem prover to prove it.

Prove it!

$ ./Simplify
> (AND (EQ (+ 0 2) 2)
  (FORALL ( x y ) (IMPLIES (EQ (+ x y) 2)
    (AND (IMPLIES (> y 0)
      (EQ (+ (+ x 1)(- y 1))
        2))
    (IMPLIES (<= y 0) (NEQ x 0))))))
1: Valid.

• Great!
• Simplify is a non-trivial five megabytes

Can We Mess Up VCGen?

• The invariant is from the user (= the adversary, the untrusted code base)
• Let’s use a loop invariant that is false, like “x \neq 0”.
• VC = \neq 0 \land
  \forall x, y. x = 0 \Rightarrow
  (y>0 \Rightarrow x+1 \neq 0 \land y\leq0 \Rightarrow x=0)
• Let’s use a loop invariant that is too weak, like “true”.
• VC = true \land
  \forall x, y. true \Rightarrow
  (y>0 \Rightarrow true \land y\leq0 \Rightarrow x=0)

Prove it!

$ ./Simplify
> (AND TRUE
  (FORALL ( x y ) (IMPLIES TRUE
    (AND (IMPLIES (> y 0) TRUE)
      (IMPLIES (<= y 0) (NEQ x 0))))))
Counterexample: context:
  (AND
    (EQ x 0)
    (<= y 0)
  )
1: Invalid.

• OK, so we won’t be fooled.

Soundness of VCGen

• Simple form
  \{ VC(c, B) \} c \{ B \}
• Or equivalently that
  \vdash VC(c, B) \Rightarrow wp(c, B)
• Proof is by induction on the structure of c
  - Try it!
• Soundness holds for any choice of the invariant!
• Next: properties and extensions of VC\xi
