Soundness and Completeness of Axiomatic Semantics
Observations

• A key part of doing research is noticing when something is incongruous. This is related to spotting patterns.
Observations

• A key part of doing research is noticing when something is incongruous. This is related to spotting patterns.

• suffix === state
• r1 r2 === c1 ; c2
• r1* === while ? do r1
• r1 | r2 === if ? then r1 else r2
What's Wrong Here?

• Look closely at this “opsem rule”

\[ s = r_1 :: r_2 :: S \]

\[ \vdash r_1 \; r_2 \text{ matches } s \text{ leaving } S \]

- string = list of characters
- regular expression = tree-structured mathematical object defined by a grammar
- syntax for list construction
- set of strings
One-Slide Summary

• A system of axiomatic semantics is sound if everything we can prove is also true: \( \text{if } \vdash \{ A \} \subset \{ B \} \text{ then } \models \{ A \} \subset \{ B \} \)

• We prove this by simultaneous 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: \( \text{if } \models \{ A \} \subset \{ B \} \text{ then } \vdash \{ A \} \subset \{ B \} \)

• Our system is relatively complete (= 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**

\[
\{A\} \subseteq \{B\} \quad \sigma \models A \quad \vdash \{A\} \Rightarrow \{B\}
\]
Soundness of Axiomatic Semantics

• Formal statement of soundness:
  \[ \text{if } \vdash \{ A \} \subseteq \{ B \} \text{ then } \models \{ A \} \subseteq \{ B \} \]
  or, equivalently
  For all \( \sigma \), if \( \sigma \vdash A \)
  and \( \text{Op} :: <c, \sigma> \downarrow \sigma' \)
  and \( \text{Pr} :: \vdash \{ A \} \subseteq \{ B \} \)
  then \( \sigma' \vdash B \)

• “Op” === “Opsem Derivation”
• “Pr” === “Axiomatic Proof”

How shall we prove this, oh class?
Not Easily!

- By induction on the structure of $c$?
  - No, problems with while and rule of consequence
- By induction on the structure of $Op$?
  - No, problems with while
- By induction on the structure of $Pr$?
  - No, problems with consequence
- By simultaneous induction on the structure of $Op$ and $Pr$
  - Yes! New Technique!
Simultaneous Induction

- Consider two structures $O_p$ and $P_r$
  - Assume that $x < y$ iff $x$ is a substructure of $y$
- Define the ordering
  $$(o, p) < (o', p')$$ iff
  $$o < o' \quad \text{or} \quad o = o' \text{ and } p < p'$$
  - Called lexicographic (dictionary) ordering
- This $<$ is a well-founded order and leads to simultaneous induction
- If $o < o'$ then $h$ can actually be larger than $h'$!
- It can even be unrelated to $h'$!
Soundness of Axiomatic Semantics

• Formal statement of **soundness**:  
  
  If \( \vDash \{ A \} \subseteq \{ B \} \) then \( \models \{ A \} \subseteq \{ B \} \)

  or, equivalently

  For all \( \sigma \), if \( \sigma \models A \)

  and \( \text{Op} :: <c, \sigma> \downarrow \sigma' \)

  and \( \text{Pr} :: \vdash \{ A \} \subseteq \{ B \} \)

  then \( \sigma' \models B \)

• “Op” = “Opsem Derivation”

• “Pr” = “Axiomatic Proof”
Simultaneous Induction

• Consider two structures Op and Pr
  - Assume that \( x < y \) iff \( x \) is a substructure of \( y \)

• Define the ordering
  \[ (o, p) \prec (o', p') \] iff
  \[ o < o' \quad \text{or} \quad o = o' \quad \text{and} \quad p < p' \]
  - Called lexicographic (dictionary) ordering

• This \( \prec \) is a **well founded order** and leads to simultaneous induction

• If \( o < o' \) then \( p \) can actually be larger than \( p' \)

• It can even be unrelated to \( p' \)!
Soundness of the While Rule
(Indiana Proof and the Slide of Doom)

• Case: last rule used in Pr : ⊢ \{A\} c \{B\} was the while rule:

\[
\begin{align*}
\text{Pr}_1 :: & \quad \vdash \{A \land b\} c \{A\} \\
\vdash \{A\} \text{ while } b \text{ do } c \{A \land \neg b\}
\end{align*}
\]

• Two possible rules for the root of Op (by inversion)
  - We’ll only do the complicated case:

\[
\begin{align*}
\text{Op}_1 :: & \quad <b, \sigma> \Downarrow \text{true} \quad \text{Op}_2 :: <c,\sigma> \Downarrow \sigma' \quad \text{Op}_3 :: <\text{while } b \text{ do } c, \sigma'> \Downarrow \sigma''
\end{align*}
\]

\[
<\text{while } b \text{ do } c, \sigma> \Downarrow \sigma''
\]

Assume that \( \sigma \models A \)

To show that \( \sigma'' \models A \land \neg b \)

• By soundness of booleans and Op\(_1\) we get \( \sigma \models b \)
  - Hence \( \sigma \models A \land b \)

• By IH on Pr\(_1\) and Op\(_2\) we get \( \sigma' \models A \)

• By IH on Pr and Op\(_3\) we get \( \sigma'' \models A \land \neg b \), q.e.d. (tricky!)
Soundness of the While Rule

- Note that in the last use of IH the derivation $Pr$ did not decrease
- But $Op_3$ was a sub-derivation of $Op$
- See Winskel, Chapter 6.5, for another example of a soundness proof
Completeness of Axiomatic Semantics

- If $\models \{A\} \subseteq \{B\}$ can we always derive $\vdash \{A\} \subset \{B\}$?
- If so, axiomatic semantics is 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 the underlying logic is complete (whenever $\models A$ we also have $\vdash A$)
  - this is called relative completeness
Examples, General Plan

• OK, so:
  \[ \vdash \{ x < 5 \land z = 2 \} y := x + 2 \{ y < 7 \} \]

• Can we prove it?
  \[ \vdash ? \{ x < 5 \land z = 2 \} y := x + 2 \{ y < 7 \} \]

• Well, we could easily prove:
  \[ \vdash \{ x+2 < 7 \} y := x + 2 \{ y < 7 \} \]

• And we know ...
  \[ \vdash x < 5 \land z = 2 \Rightarrow x+2 < 7 \]

• Shouldn’t those two proofs be enough?
Proof Idea

- Dijkstra’s idea: To verify that \{ A \} \subseteq \{ B \}
  a) Find out all predicates \( A' \) such that \( \vdash \{ A' \} \subseteq \{ B \} \)
    - call this set \( \text{Pre}(c, B) \) (Pre = “pre-conditions”)
  b) Verify for one \( A' \in \text{Pre}(c, B) \) that \( A \Rightarrow A' \)

- Assertions can be ordered:
  \[
  \text{false} \quad \Rightarrow \quad \text{true}
  \]

  \[
  \begin{array}{c}
  \text{strong} \\
  \text{A} \\
  \frac{}{\text{weakest precondition: WP}(c, B)} \text{ weak}
  \end{array}
  \]

- Thus: compute \( \text{WP}(c, B) \) and prove \( A \Rightarrow \text{WP}(c, B) \)
Proof Idea (Cont.)

- **Completeness** of axiomatic semantics:
  \[ \text{If } \models \{ A \} \subset \{ B \} \text{ then } \vdash \{ A \} \subset \{ B \} \]

- Assuming that we can compute \( \text{wp}(c, B) \) with the following properties:
  - \( \text{wp} \) is a precondition (according to the Hoare rules)
    \[ \vdash \{ \text{wp}(c, B) \} \subset \{ B \} \]
  - \( \text{wp} \) is (truly) the weakest precondition
    \[ \text{If } \models \{ A \} \subset \{ B \} \text{ then } \models A \Rightarrow \text{wp}(c, B) \]
    \[ \vdash A \Rightarrow \text{wp}(c, B) \quad \vdash \{ \text{wp}(c, B) \} \subset \{ B \} \]
    \[ \vdash \{ A \} \subset \{ B \} \]

- We also need that whenever \( \models A \) then \( \vdash A \)!
Q: Bonus

- Despite having physically appeared in only about ten movies, this Indian singer has received the *Bharat Ratna* (India's highest civilian honor) and holds the Guinness Book of World Records entry for “most recordings” (30,000 songs by 1987). At one point the Pakistani prime minister said the he would “gladly exchange [her] for Kashmir”. She is the sister of Asha Bhosle and specializes in “playback” or “voiceover” movie music.
Q: Writing

• "Some coffee, Mr. Covenant?"

• "No!" he panted, glaring. The gelid liquid was anthraciously black, atramentous, nigrescent as his carious and macerated soul. "No," he groaned. "Do you hear? I will not!" Shaking, he fumbled for his empty mug, clawing at it with numb hands like blocks of rotted wood. Finally, gasping, he closed his fingers on the malefic vessel, upending it, then ramming it downward to the table again... violently stopping the irrefragable, ineluctable maw with intransigent formica. The sudden whipcrack sound threw a refulgent oriflamme of pain across his sight, and he closed his eyes with a febrile shudder. "No," he whispered. No more. No more.

• "All righty then, I'll be right back with your check!"
Q: Writing

• “In order to X”
  - “To X”

• “By induction on the hypothesis”
  - “By the induction hypothesis”

• “Choose X at random”
  - “Let X be arbitrary”

• “For the next step in the proof to proceed, set the value of x to be 2.”
  - “Let x be 2.”
Axiomatic Semantics: Preconditions

My students drew me into another political argument.

Eh; it happens.

Lately, political debates bother me. They just show how good smart people are at rationalizing.

The world is so complicated - the more I learn, the less clear anything gets. There are too many ideas and arguments to pick and choose from. How can I trust myself to know the truth about anything?

And if everything I know is so shaky, what on earth am I doing teaching?

I guess you just do your best. No one can impart perfect universal truths to their students.

*Ahem*

... except math teachers.

Thank you.
Weakest Preconditions

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

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

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

  - $wp(if E then c_1 else c_2, B) = \{ E \Rightarrow A_1 \land \neg E \Rightarrow A_2 \} if E then c_1 else c_2 \{B\}$

- $wp(if E then c_1 else c_2, B) = E \Rightarrow wp(c_1, B) \land \neg E \Rightarrow wp(c_2, B)$
Weakest Preconditions for Loops

• We start from the \textit{unwinding} equivalence
  
  \[
  \text{while } b \text{ do } c \quad = \quad \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ else skip}
  \]

• Let \( w = \text{while} b \text{ do } c \) and \( W = \text{wp}(w, B) \)

• We have that
  
  \[
  W = b \Rightarrow \text{wp}(c, W) \quad \land \quad \neg b \Rightarrow B
  \]

• But this is a \textit{recursive equation}!
  - Mathematicians solve these using domain theory

• But we need a domain for assertions
  - This will give us a way to define “weakest”
A Partial Order for Assertions

• Which assertion contains the least information?
  - “true”: it does not say anything about the state

• What is an appropriate information ordering?
  \[ A \subseteq A' \iff \models A' \Rightarrow A \]

• Is this partial order complete?
  - Take a chain \(A_1 \subseteq A_2 \subseteq \ldots\)
  - Let \(\bigwedge A_i\) be the infinite conjunction of \(A_i\)
    \[\sigma \models \bigwedge A_i \iff \text{for all } i \text{ we have that } \sigma \models A_i\]
  - I assert that \(\bigwedge A_i\) is the least upper bound

• Can \(\bigwedge A_i\) be expressed finitely in our language of assertions?
  - In many cases: yes (see Winskel), we’ll assume yes for now
Weakest Precondition for WHILE

• Use the fixed-point theorem
  \[ F(A) = b \Rightarrow \wp(c, A) \land \neg b \Rightarrow B \]
  - (Where did this come from? Two slides back!)
  - I assert that F is both monotonic and continuous

• The least-fixed point (= the weakest fixed point) is
  \[ \wp(w, B) = \bigwedge F^i(\text{true}) \]

• (Notice that we are not working on a flat domain. Bonus: What does that sentence mean?)
Weakest Preconditions (Cont.)

• Define a family of wp’s
  - \(wp_k(\text{while } e \text{ do } c, B)\) = weakest precondition on which
    the loop terminates in \(B\) if it terminates in \(k\) or fewer
    iterations
    \(wp_0 = \neg E \Rightarrow B\)
    \(wp_1 = E \Rightarrow wp(c, wp_0) \land \neg E \Rightarrow B\)
    ...
  - \(wp(\text{while } e \text{ do } c, B) = \bigwedge_{k \geq 0} wp_k = \text{lub} \{wp_k \mid k \geq 0\}\)
  - See Necula document 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?
Not Quite Weakest Preconditions

• Recall what we are trying to do:

\[
\text{false} \quad \Rightarrow \quad \text{true}
\]

\[
\text{Pre}(s, B)
\]

(weakest precondition: \( WP(c, B) \))

(verification condition: \( VC(c, B) \))

• Construct a **verification condition**: \( VC(c, B) \)
  
  - Our loops will be annotated with loop invariants!
  
  - \( VC \) is guaranteed to be stronger than \( WP \)
  
  - But still weaker than \( A \): \( A \Rightarrow VC(c, B) \Rightarrow WP(c, B) \)
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:
  \[
  \text{while}_{Inv} \ b \ \text{do } \ c
  \]
  - Invariant formula $Inv$ must hold every time before $b$ is evaluated
- A process for computing VC(annotated_command, post_condition) is called VCGen
Verification Condition Generation

- Mostly follows the definition of the wp function:

  \[ VC(\text{skip}, B) = B \]

  \[ VC(c_1; c_2, B) = VC(c_1, VC(c_2, B)) \]

  \[ VC(\text{if } b \text{ then } c_1 \text{ else } c_2, B) = \]

  \[ b \Rightarrow VC(c_1, B) \land \neg b \Rightarrow VC(c_2, B) \]

  \[ VC(x := e, B) = [e/x] B \]

  \[ VC(\text{let } x = e \text{ in } c, B) = [e/x] VC(c, B) \]

  \[ VC(\text{while}_{\text{inv}} b \text{ do } c, B) = ? \]
VCGen for WHILE

\[
\text{VC}(\text{while}_{\text{Inv}} \ e \ do \ c, \ B) =
\begin{align*}
\text{Inv} & \land (\forall x_1 \ldots x_n. \ \text{Inv} \Rightarrow (e \Rightarrow \text{VC}(c, \ \text{Inv}) \land \neg e \Rightarrow B) )
\end{align*}
\]

- **Inv** is the loop invariant (provided externally)
- \(x_1, \ldots, x_n\) are all the variables modified in \(c\)
- The \(\forall\) is similar to the \(\forall\) in mathematical induction:
  \[
P(0) \land \forall n \in \mathbb{N}. \ P(n) \Rightarrow P(n+1)
  \]
Example VCGen Problem

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

```c
x = 0;
y = 2;
while $x + y = 2$, $y > 0$ do
    y := y - 1;
    x := x + 1
```

First, what do we expect? What pre-condition do we need to ensure $x \neq 0$ after this?
Example of VC

- By the sequencing rule, first we do the while loop (call it $w$):

  ```
  while $x+y=2 \land y > 0$ do
    y := y - 1;
    x := x + 1
  
  VCGen(w, x \neq 0) = x+y=2 \land
  \forall x,y. x+y=2 \Rightarrow (y>0 \Rightarrow VC(c, x+y=2) \land y\leq 0 \Rightarrow x \neq 0)
  
  VCGen(y:=y-1 ; x:=x+1, x+y=2) =
  (x+1) + (y-1) = 2
  
  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 \neq 0)
  ```
Example of VC (2)

• \( \text{VC}(w, x \neq 0) = x+y=2 \land \)
  \( \forall x,y. \ x+y=2 \Rightarrow \)
  \( (y>0 \Rightarrow (x+1)+(y-1)=2 \land y \leq 0 \Rightarrow x \neq 0) \)

• \( \text{VC}(x := 0; y := 2 ; w, x \neq 0) = 0+2=2 \land \)
  \( \forall x,y. \ x+y=2 \Rightarrow \)
  \( (y>0 \Rightarrow (x+1)+(y-1)=2 \land y \leq 0 \Rightarrow x \neq 0) \)

• So now we ask an automated theorem prover to prove it.
Thoreau, Thoreau, Thoreau

$ ./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.

• Huzzah!
• Simplify is a non-trivial five megabytes
• Z3 is 15+ 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 too weak, like “true”.
  \[ VC = \text{true} \land \forall x, y. \text{true} \Rightarrow (y > 0 \Rightarrow \text{true} \land y \leq 0 \Rightarrow x \neq 0) \]
- Let’s use a loop invariant that is false, like “\(x \neq 0\)”.
  \[ VC = 0 \neq 0 \land \forall x, y. x \neq 0 \Rightarrow (y > 0 \Rightarrow x + 1 \neq 0 \land y \leq 0 \Rightarrow x \neq 0) \]
$ ./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
  \[ \vdash \{ \text{VC}(c, B) \} c \{ B \} \]

- Or equivalently that
  \[ \vdash \text{VC}(c, B) \Rightarrow \text{wp}(c, B) \]

- Proof is by induction on the structure of \( c \)
  - Try it!

- Soundness holds for any choice of invariant!

- Next: properties and extensions of VCs
Questions

- Homework?
- Project proposal?