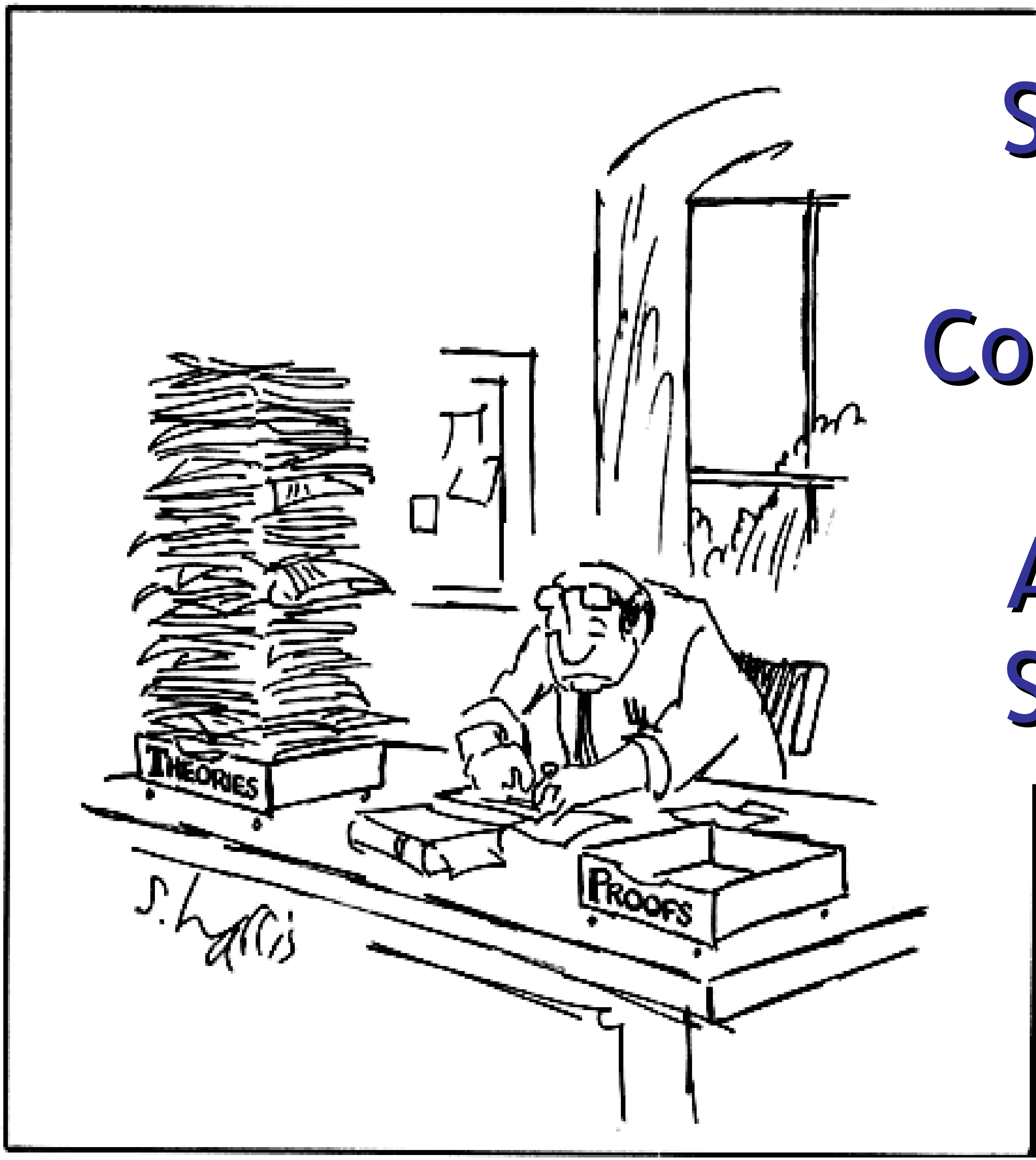


Soundness and Completeness of Axiomatic Semantics

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”

string = list of characters

syntax for list construction

$$\frac{s = r1 :: r2 :: S}{\vdash r1\ r2\ \textbf{matches}\ s\ \textbf{leaving}\ S}$$

regular expression = tree-structured mathematical object defined by a grammar

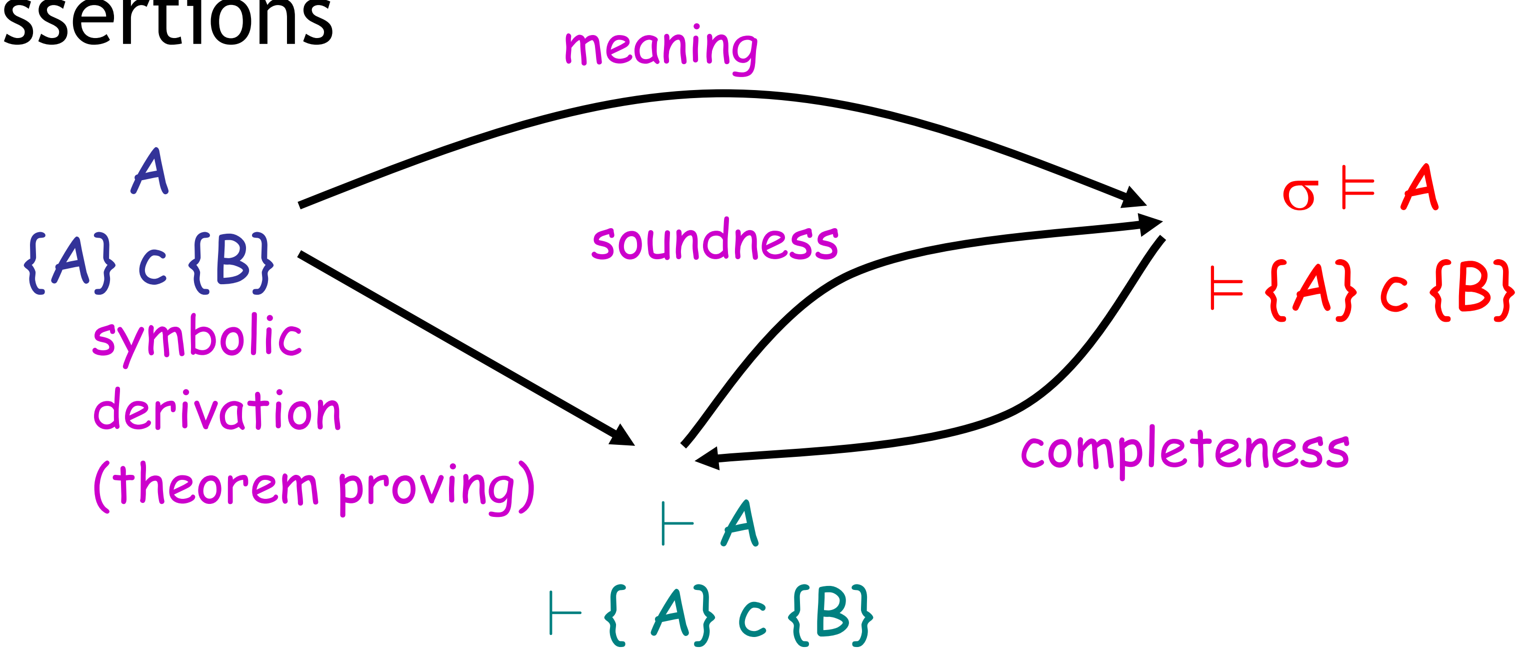
set of strings

One-Slide Summary

- A system of axiomatic semantics is **sound** if everything we can prove is also true: if $\vdash \{ A \}\ c\ \{ B \}$ then $\vDash \{ A \}\ c\ \{ B \}$
- 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: if $\vDash \{ A \}\ c\ \{ B \}$ then $\vdash \{ A \}\ c\ \{ B \}$
- 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



Soundness of Axiomatic Semantics

- Formal statement of soundness:

  if $\vdash \{ A \}\ c\ \{ B \}$ then $\models \{ A \}\ c\ \{ B \}$

  or, equivalently

  For all σ, if $\sigma \models A$

  and Op :: $<c, \sigma> \Downarrow \sigma'$

  and Pr :: $\vdash \{ A \}\ c\ \{ B \}$

  then $\sigma' \models B$

How shall we prove this, oh class?

- "Op" === "Opsem Derivation"
- "Pr" === "Axiomatic Proof"

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 Op and Pr
  - Assume that $x < y$ iff x is a substructure of y
- Define the ordering

$$(o, p) \prec (o', p') \text{ iff}$$

$$o < o' \quad \text{or} \quad o = o' \text{ and } p < p'$$

  - Called lexicographic (dictionary) ordering
- This $\prec$ 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 $\vdash \{ A \}\ c\ \{ B \}$ then $\models \{ A \}\ c\ \{ B \}$

  or, equivalently

  For all σ, if $\sigma \models A$

  and Op :: $\langle c, \sigma \rangle \Downarrow \sigma'$

  and Pr :: $\vdash \{ A \}\ c\ \{ B \}$

  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

  $o < o'$ or $o = o'$ and $p < p'$
  - 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 : $\vdash$ {A} c {B} was the while rule:

$$\frac{Pr_1 :: \vdash \{A \wedge b\}\ c\ \{A\}}{\vdash \{A\}\ \text{while } b \text{ do } c\ \{A \wedge \neg\, b\}}$$

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

$$\frac{Op_1 :: \langle b, \sigma\rangle \Downarrow \text{true} \qquad Op_2 :: \langle c,\sigma\rangle \Downarrow \sigma' \qquad Op_3 :: \langle \text{while } b \text{ do } c,\ \sigma'\rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c,\ \sigma\rangle \Downarrow \sigma''}$$

Assume that $\sigma \vDash A$

To show that $\sigma'' \vDash A \wedge \neg\, b$

- By soundness of booleans and Op_1 we get $\sigma \vDash b$
  - Hence $\sigma \vDash A \wedge b$
- By IH on Pr_1 and Op_2 we get $\sigma' \vDash A$
- By IH on Pr and Op_3 we get $\sigma'' \vDash A \wedge \neg\, b$, 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 $\models \{A\}\ c\ \{B\}$ can we always derive $\vdash \{A\}\ c\ \{B\}$?
- 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:

$$\models \{ x < 5 \wedge z = 2 \}\ y := x + 2\ \{ y < 7 \}$$

- Can we prove it?

$$?\vdash? \{ x < 5 \wedge z = 2 \}\ y := x + 2\ \{ y < 7 \}$$

- Well, we *could* easily prove:

$$\vdash \{ x+2 < 7 \}\ y := x + 2\ \{ y < 7 \}$$

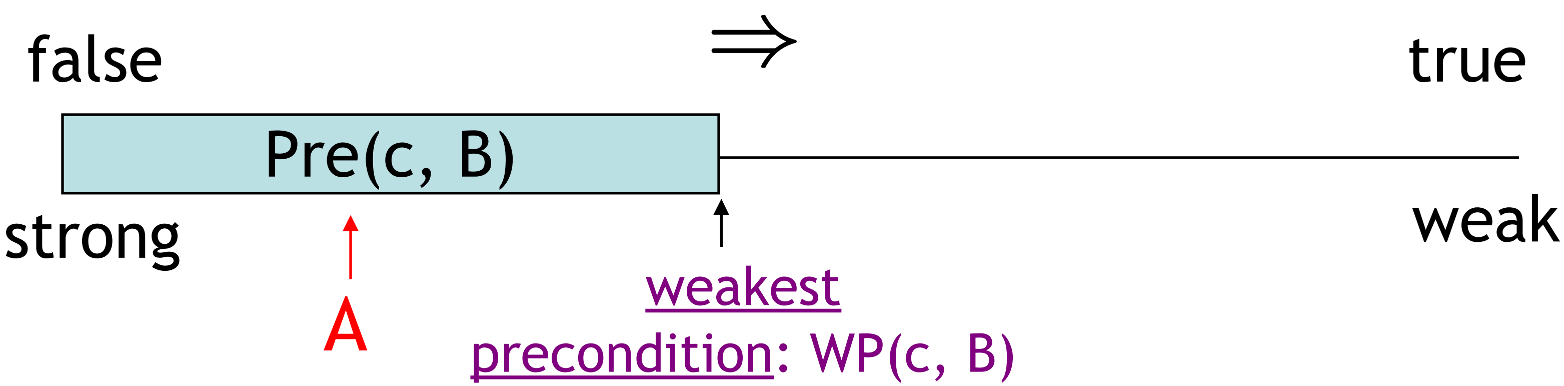
- And we know …

$$\vdash x < 5 \wedge z = 2 \Rightarrow x+2 < 7$$

- Shouldn't those two proofs be enough?

Proof Idea

- Dijkstra's idea: To verify that { A } c { B }
  a) Find out all predicates A' such that $\models$ { A' } c { B }
    - call this set Pre(c, B)   (Pre = "pre-conditions")
  b) Verify for one A' $\in$ Pre(c, B) that A $\Rightarrow$ A'
- Assertions can be ordered:



- Thus: compute WP(c, B) and prove A $\Rightarrow$ WP(c, B)

Proof Idea (Cont.)

- 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:
  - wp is a precondition (according to the Hoare rules)

    $\vdash \{ wp(c, B) \}\ c\ \{ B \}$
  - wp is *(truly)* the weakest precondition

    If $\models \{ A \}\ c\ \{ B \}$ then $\models A \Rightarrow wp(c, B)$

$$\frac{\vdash A \Rightarrow wp(c, B) \qquad \vdash \{wp(c, B)\}\ c\ \{B\}}{\vdash \{A\}\ c\ \{B\}}$$

- 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.

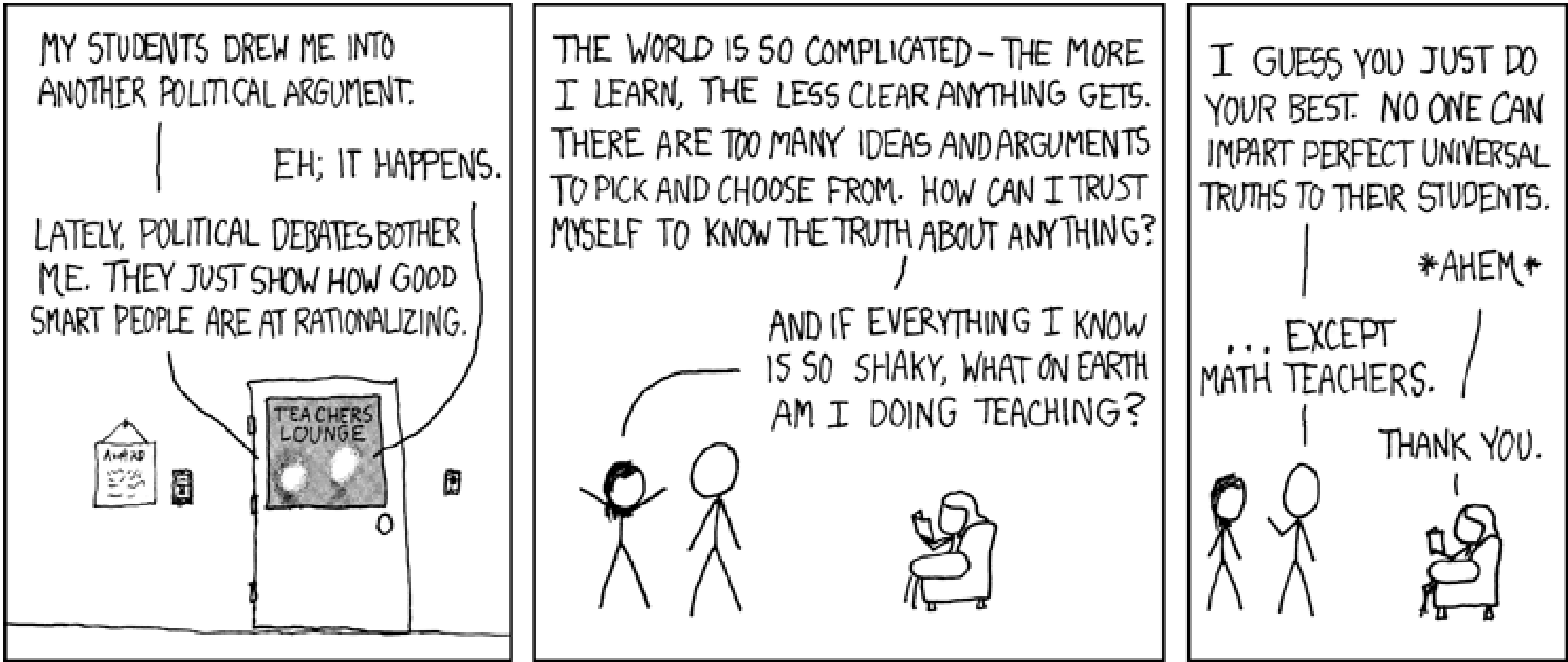
Q: Writing

- "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



Weakest Preconditions

- Define wp(c, B) inductively on c, following the Hoare rules:
- $wp(c_1; c_2, B) =$
  $wp(c_1, wp(c_2, B))$

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

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

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

$$\frac{\{A_1\}\ c_1\ \{B\} \qquad \{A_2\}\ c_2\ \{B\}}{\{\ E \Rightarrow A_1 \wedge \neg\ E \Rightarrow A_2\}\ \text{if } E \text{ then } c_1 \text{ else } c_2\ \{B\}}$$

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

Weakest Preconditions for Loops

- We start from the *unwinding* equivalence

  while b do c   =

    if b then c; while b do c else skip
- Let   w = while b do c   and   W = wp(w, B)
- We have that

  $$W = b \Rightarrow wp(c, W) \quad \wedge \quad \neg\, b \Rightarrow B$$
- But this is a 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 ?

$$A \sqsubseteq A' \quad \text{iff} \quad \models A' \Rightarrow A$$

- 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

$$\sigma \models \bigwedge A_i \ \text{ iff for all i we have that } \sigma \models A_i$$

  - 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

$$F(A) = b \Rightarrow wp(c, A) \wedge \neg b \Rightarrow B$$

  - (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(while e do c, B)** = weakest precondition on which the loop terminates in B if it terminates in k or fewer iterations

$wp_0 = \neg E \Rightarrow B$

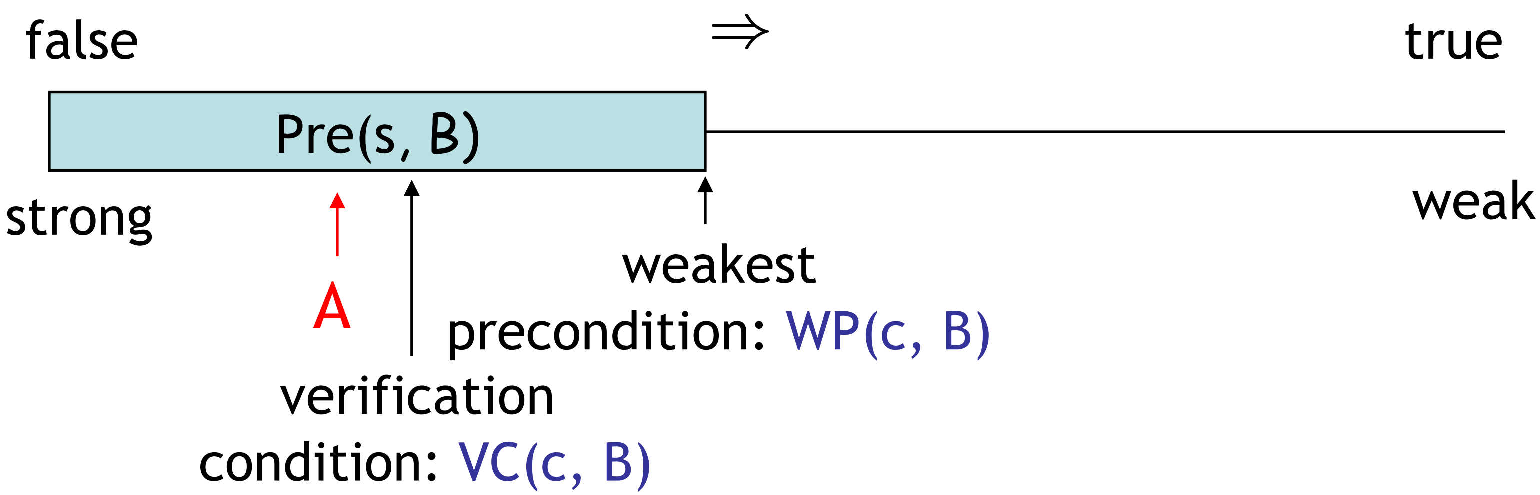
$wp_1 = E \Rightarrow wp(c, wp_0) \wedge \neg E \Rightarrow B$

…

- wp(while e do c, B) = $\bigwedge_{k \geq 0} wp_k$ = 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:



- 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:

$$\text{while}_{\text{Inv}} \text{ b do c}$$

  - 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:

$$VC(\text{skip}, B) = B$$

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

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

$$b \Rightarrow VC(c_1, B) \wedge \neg b \Rightarrow VC(c_2, B)$$

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

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

$$VC(\text{while}_{Inv}\ b \text{ do } c, B) = ?$$

VCGen for WHILE

$\mathrm{VC}(\mathrm{while}_{\mathrm{Inv}}\ e\ \mathrm{do}\ c,\ B) =$

$$\underbrace{\mathrm{Inv}}_{\text{Inv holds on entry}} \wedge (\forall x_1 \ldots x_n.\ \mathrm{Inv} \Rightarrow \underbrace{(e \Rightarrow \mathrm{VC}(c, \mathrm{Inv})}_{\text{Inv is preserved in an arbitrary iteration}} \wedge \underbrace{\neg e \Rightarrow B}_{\text{B holds when the loop terminates in an arbitrary iteration}}))$$

- 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:

$$P(0) \wedge \forall n \in \mathbb{N}.\ P(n) \Rightarrow P(n+1)$$

Example VCGen Problem

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

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):

  $\text{while}_{x+y=2}\ y > 0$ do

  $y := y - 1;$

  $x := x + 1$

Preserve loop invariant

Ensure post on exit

- VCGen(w, $x \neq 0$) = $x+y=2 \wedge$

  $\forall x,y.\ x+y=2 \Rightarrow (y>0 \Rightarrow \text{VC}(c,\ x+y=2)\ \wedge\ y\leq 0 \Rightarrow x \neq 0)$
- VCGen($y:=y-1$; $x:=x+1$, $x+y=2$) =

  $(x+1) + (y-1) = 2$
- w Result: $x+y=2 \wedge$

  $\forall x,y.\ x+y=2 \Rightarrow (y>0 \Rightarrow (x+1)+(y-1)=2\ \wedge\ y\leq 0 \Rightarrow x \neq 0)$

Example of VC (2)

- $VC(w, x \neq 0) = x+y=2 \wedge$

  $\forall x,y.\ x+y=2 \Rightarrow$

  $(y>0 \Rightarrow (x+1)+(y-1)=2 \ \wedge\ y \leq 0 \Rightarrow x \neq 0)$

- $VC(x := 0;\ y := 2\ ;\ w,\ x \neq 0) = 0+2=2 \wedge$

  $\forall x,y.\ x+y=2 \Rightarrow$

  $(y>0 \Rightarrow (x+1)+(y-1)=2 \ \wedge\ y \leq 0 \Rightarrow x \neq 0)$

- 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 = $\text{true} \wedge \quad \forall x,y.\ \text{true} \Rightarrow$

  $(y>0 \Rightarrow \text{true} \ \wedge\ y \leq 0 \Rightarrow x \neq 0)$
- Let's use a loop invariant that is false, like "$x \neq 0$".
- VC = $0 \neq 0 \wedge \quad \forall x,y.\ x \neq 0 \Rightarrow$

  $(y>0 \Rightarrow x+1 \neq 0 \ \wedge\ y \leq 0 \Rightarrow x \neq 0)$

Emerson, Emerson, Emerson

```
$ ./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

$$\models \{ VC(c,B) \}\ c\ \{ B \}$$

- Or equivalently that

$$\models VC(c, B) \Rightarrow wp(c, B)$$

- 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?