

Theorem Proving

Review

- Hoare Triples
 - Partial correctness: $\{P\} s \{Q\}$
 - Total correctness: $[P] s [Q]$
- Weakest preconditions
 - To verify $\{P\} s \{Q\}$
 - compute $wp(s, Q)$ and prove $P \Rightarrow wp(s, Q)$
 - hard for loops
- Verification conditions
 - like wp , but use invariants for loops

Weakest Preconditions: Example

- $wp(x:=x+1, x=y)$
= $x+1=y$
- $wp(y:=y+1; x:=x+1, x=y)$
= $wp(y:=y+1, wp(x:=x+1, x=y))$
= $wp(y:=y+1, x+1=y)$
= $(x+1=y+1)$
= $(x=y)$

Weakest Preconditions For Loops

• $wp(\text{while } E \text{ do } S, Q)$

$$= \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, wp(\text{while } E \text{ do } S, Q))$$

$$= \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, wp(\text{while } E \text{ do } S, Q)))$$

$$= \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, \neg E \Rightarrow Q$$

$$\text{K}_y E \Rightarrow wp(S, wp(\text{while } E \text{ do } S, Q))))$$

... ad infinitum ... hard to compute

Verification Condition Generation

- $vc(\text{while}_I E \text{ do } s, Q) =$

$$\underbrace{I}_{\text{I holds on entry}} \wedge \underbrace{(\exists x_1 \dots x_n. I \wedge (E \wedge vc(s, I) \wedge \neg Q))}_{\text{I is preserved in an arbitrary iteration}} \wedge \underbrace{\neg Q}_{\text{Q holds when the loop terminates}}$$

- I is the loop invariant (provided externally)
- x_1, \dots, x_n are all the variables modified in s
- Definition says:
 - the invariant holds initially,
 - and on any loop iteration where the invariant initially holds
 - if the loop terminates then the postcondition holds
 - and if the loop does not terminate, then after s , the invariant holds

Weakest Precondition Generation

$$\text{wp}(s_1; s_2, R) = \text{wp}(s_1, \text{wp}(s_2, R))$$

$$\text{wp}(x := E, Q) = Q[E/x]$$

$$\text{wp}(\text{if } E \text{ then } s_1 \text{ else } s_2, Q) = (E \wedge \text{wp}(s_1, Q)) \vee (\neg E \wedge \text{wp}(s_2, Q))$$

$$\text{wp}(\text{while } E \text{ do } S, Q) = \dots \dots \text{hard}$$

Verification Condition Generation

$$vc(s_1; s_2, R) = vc(s_1, vc(s_2, R))$$

$$vc(x := E, Q) = Q[E/x]$$

$$vc(\text{if } E \text{ then } s_1 \text{ else } s_2, Q) = (E \neq vc(s_1, Q)) \wedge (E \neq vc(s_2, Q))$$

$$vc(\text{while}_I E \text{ do } s, Q) =$$

$$I \wedge (\exists x_1 \dots x_n. I \neq (E \neq VC(s, I) \wedge E \neq Q))$$

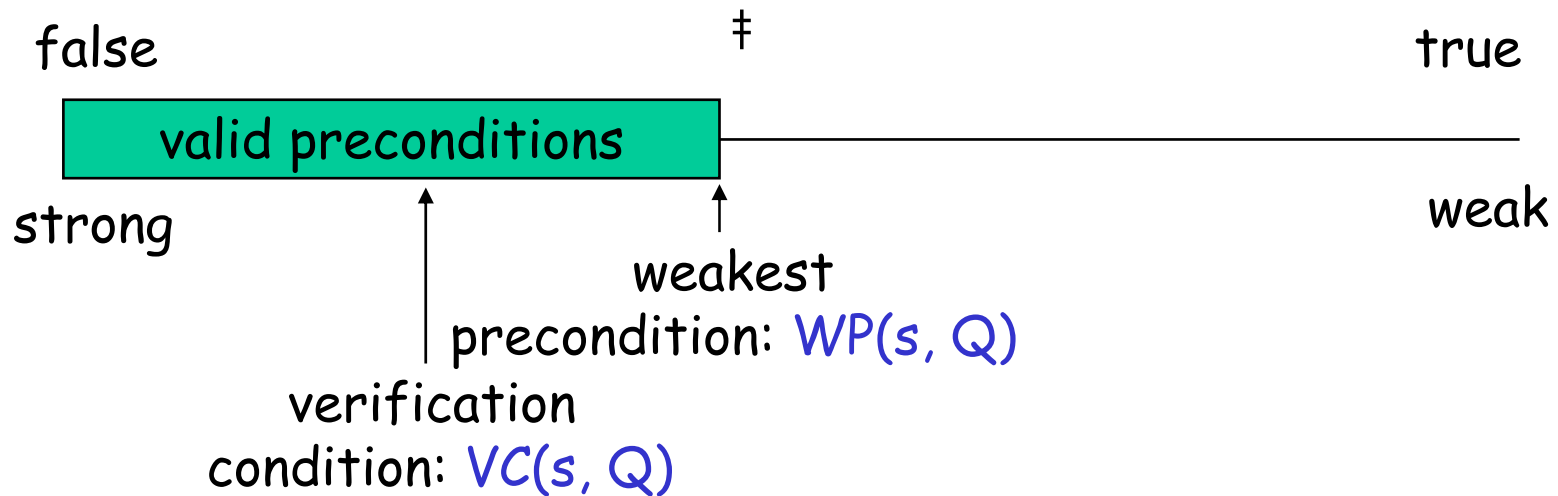
Verification Conditions: Example

- What is $vc(\text{while}_I x>0 \text{ do } x:=x-1, x=0)$
- Depends on I

- Suppose $I = \text{false}$
- Suppose $I = (x<0)$
- Suppose $I = (x>0 \text{ and } x \in \text{Integer})$

VCs are less weak than WPs

- Recall what we are trying to do:



What about Exceptions?

- $s ::=$
 - $x := E$
 - if E then s else s
 - $s ; s$
 - while E do e
 - throw*
 - try s catch s*
- Statements may terminate *normally* or *exceptionally*
- $wp(s, Q, R)$ = set of states from which
 - s may terminate normally in a state satisfying Q , or
 - s may terminate exceptionally in a state satisfying R

Computing WP for Exceptions

$$\text{wp}(x := E, Q, R) = Q[E/x]$$

$$\begin{aligned} \text{wp}(\text{if } E \text{ then } s_1 \text{ else } s_2, Q, R) \\ = (E \dagger \text{wp}(s_1, Q, R)) \\ \wedge (\neg E \dagger \text{wp}(s_2, Q, R)) \end{aligned}$$

$$\text{wp}(\text{throw}, Q, R) =$$

$$\text{wp}(s_1; s_2, Q, R) = \text{wp}(s_1, \quad , \quad)$$

$$\begin{aligned} \text{wp}(\text{try } s_1 \text{ catch } s_2, Q, R) \\ = \text{wp}(s_1, \quad , \quad) \end{aligned}$$

Computing WP for Exceptions

$$\text{wp}(x := E, Q, R) = Q[E/x]$$

$$\begin{aligned} \text{wp}(\text{if } E \text{ then } s_1 \text{ else } s_2, Q, R) \\ = (E \dagger \text{wp}(s_1, Q, R)) \\ \wedge (\neg E \dagger \text{wp}(s_2, Q, R)) \end{aligned}$$

$$\text{wp}(\text{throw}, Q, R) = R$$

$$\text{wp}(s_1; s_2, Q, R) = \text{wp}(s_1, \text{wp}(s_2, Q, R), R)$$

$$\begin{aligned} \text{wp}(\text{try } s_1 \text{ catch } s_2, Q, R) \\ = \text{wp}(s_1, Q, \text{wp}(s_2, Q, R)) \end{aligned}$$

VCs for procedures

- Consider

requires true

ensures result > 0

```
void abs(int x) {
```

```
  if (x<0)
```

```
    then result := -x
```

```
    else result := x
```

```
}
```

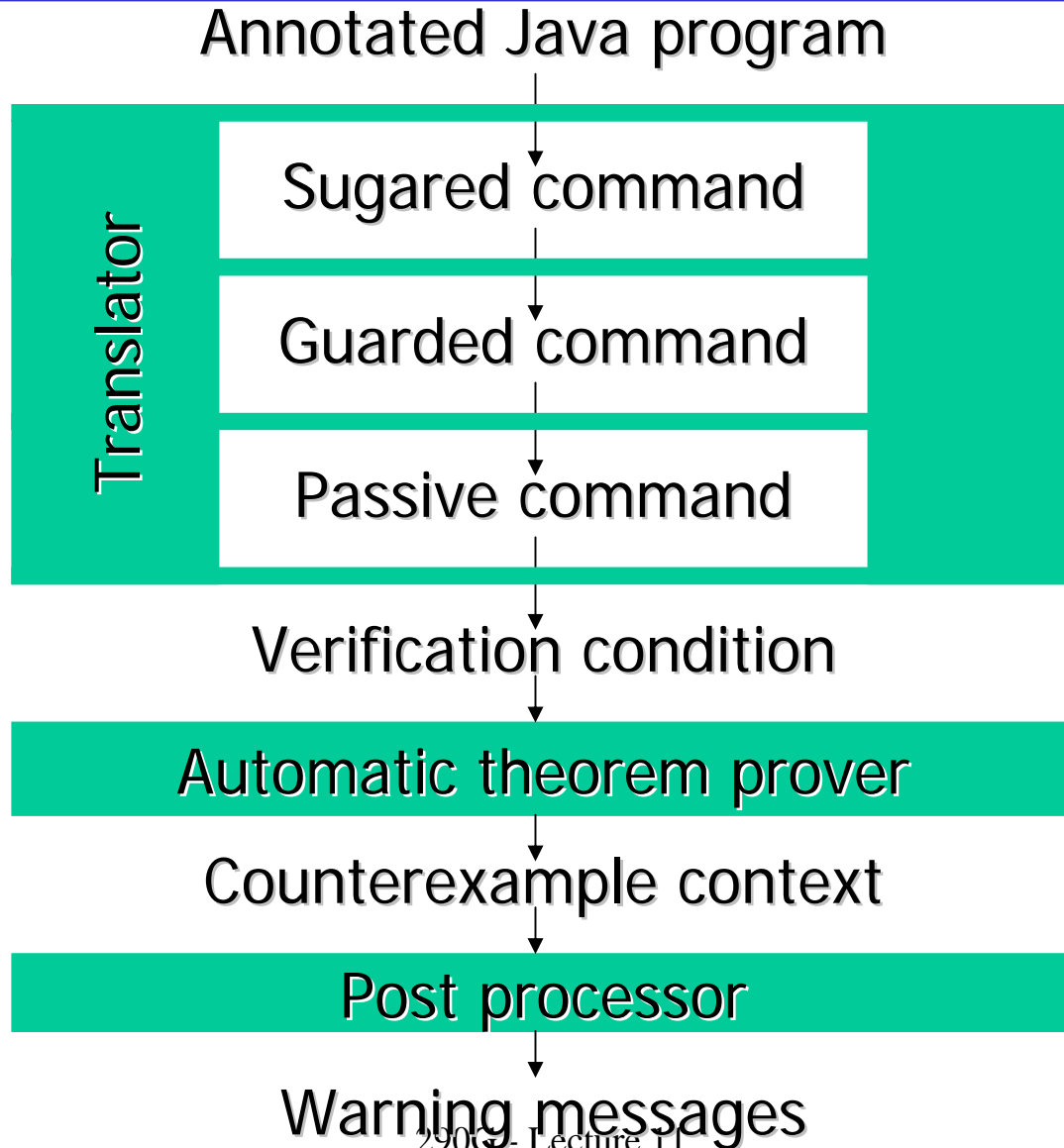
- VC is

precondition \Rightarrow vc(body, postcondition)

true \Rightarrow vc(body, result > 0)

true \Rightarrow ((x<0 \Rightarrow -x>0) \wedge (x>0 \Rightarrow x>0))

ESC/Java architecture



(AND

(<: T_T |T_java.lang.Object|)

(EQ T_T (asChild T_T |T_java.lang.Object|))

(DISTINCT arrayType |T_boolean| |T_char| |T_byte| |T_short| |T_int|

|T_long| |T_float| |T_double| |T_.TYPE|

T_T |T_java.lang.Object|)))

(EXPLIES

(LBNNEG |x:T_int| abs(x) {

(IMPLIES

if (x < 0) { x = -x; }

(AND

// @assert x >= 0;

(EQ |elems@pre| elems)

(EQ elems (asElms elems))

(< (eClosedTime elems) alloc)

(EQ LS (asLockSet LS))

(EQ |alloc@pre| alloc))

(NOT

(AND

(EQ |@true| (is |x:2.21| T_int))

(OR

(AND

(OR

(AND

(< |x:2.21| 0)

(LBNPOS |trace.Then^0,3.15| (EQ |@true| |@true|))

(EQ |x:3.17| (- 0 |x:2.21|))) 11

Annotated Java program

Sugared command

Primitive command

Passive command

Translator

Verification condition

Automatic theorem prover

Counterexample context

Post processor

Warning messages

Deciding Validity of VCs

- VC is
 $\text{true} \Rightarrow ((x < 0 \Rightarrow -x > 0) \wedge (x < 0 \Rightarrow x > 0))$
- Is VC valid?
- Is it true for *all* values of x ?
- Error condition is negation of the VC
 $\neg(\text{true} \Rightarrow ((x < 0 \Rightarrow -x > 0) \wedge (x < 0 \Rightarrow x > 0)))$
- Is EC satisfiable?
- Is EC true for *any* values of x ?
- If so, then VC is false for that x , and so is invalid
- That x is a counter-example

Deciding Satisfiability of ECs

- A hard (but solvable) problem ...
- Start with a simpler problem ...
- Satisfiability of boolean formula (SAT)
 - canonical NP-complete problem
 - rapid progress in last few years
 - many applications for SAT solvers
 - including in theorem proving

Boolean Formulas (CNF)

- variable v
- literal $l ::= v \mid \neg v$
- clause $c ::= l_1 \vee \dots \vee l_n$
- clause set $s ::= c_1 \wedge \dots \wedge c_n$

Davis-Putnam Algorithm

- variable v
- literal $l ::= v \mid \neg v \dots$
- clause $c ::= l_1 \vee \dots \vee l_n$
- clause set $s ::= c_1 \wedge \dots \wedge c_n$

- Rules
 - [Done] If a clause is empty
 - then clause set is unsatisfiable
 - [BCP] If a unit clause
 - then assign that literal true
 - [Split] Pick literal. Try assigning it true, and then try assigning its negation true
 - To assign a literal true
 - remove clauses with that literal
 - remove negation of literal from other clauses

Davis-Putnam Algorithm (cont)

- Example: (each line a separate clause)

a b c

a -b

b c

a c

-a -b

-c a

Next week

- Leverage SAT to decide satisfiability of ECs
- Verifun paper gives overview of this approach
 - NASA attendees: read sections 1+2 and no review.

Later today

- Arnaud Venet and Guillaume Brat on NASA's *C Global Surveyor* in Crown College, room 105, 2-3:45pm.
- Arnaud Venet
 - Rapid Inference of Interprocedural Numerical Invariants for Large C Programs by Abstract Interpretation
- Guillaume Brat
 - Software Analysis Opportunities at NASA

Presentations

- Harry: Feb 5
- Dorrit: Feb 24
- Min: March 11

- there's still room for you ...

How am I doing?
