

# Concrete Semantics

with Isabelle/HOL

Peter Lammich (slides adopted from Tobias Nipkow)

Fakultät für Informatik  
Technische Universität München

2018-11-4



# Part II

## Semantics

# Chapter 7

IMP:

A Simple Imperative Language

- ① IMP Commands
- ② Big-Step Semantics
- ③ Small-Step Semantics

① IMP Commands

② Big-Step Semantics

③ Small-Step Semantics

# Terminology

**Statement:** declaration of fact or claim

*Semantics is easy.*

**Command:** order to do something

*Study the book until you have understood it.*

Expressions are *evaluated*, commands are *executed*

# Commands

Concrete syntax:

$$\begin{aligned} com &::= \text{SKIP} \\ &| \text{string} ::= aexp \\ &| com \ ; \ ; \ com \\ &| \text{IF } bexp \ \text{THEN } com \ \text{ELSE } com \\ &| \text{WHILE } bexp \ \text{DO } com \end{aligned}$$

# Commands

Abstract syntax:

**datatype** *com* = *SKIP*  
| *Assign string aexp*  
| *Seq com com*  
| *If bexp com com*  
| *While bexp com*



Com.thy

① IMP Commands

② Big-Step Semantics

③ Small-Step Semantics

# Big-step semantics

Concrete syntax:

$$(com, initial-state) \Rightarrow final-state$$

Intended meaning of  $(c, s) \Rightarrow t$ :



Command  $c$  started in state  $s$  terminates in state  $t$

“ $\Rightarrow$ ” here not type!

# Big-step rules

$$(SKIP, s) \Rightarrow s$$

$$(x ::= a, s) \Rightarrow s(x := \text{aval } a \ s)$$

$$\frac{(c_1, s_1) \Rightarrow s_2 \quad (c_2, s_2) \Rightarrow s_3}{(c_1;; c_2, s_1) \Rightarrow s_3}$$

## Big-step rules

$$\frac{bval\ b\ s \quad (c_1, s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \Rightarrow t}$$

$$\frac{\neg\ bval\ b\ s \quad (c_2, s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \Rightarrow t}$$

# Big-step rules

$$\frac{\neg \text{bval } b \ s}{(\text{WHILE } b \text{ DO } c, s) \Rightarrow s}$$

$$\frac{\begin{array}{c} \text{bval } b \ s_1 \\ (c, s_1) \Rightarrow s_2 \end{array} \quad (\text{WHILE } b \text{ DO } c, s_2) \Rightarrow s_3}{(\text{WHILE } b \text{ DO } c, s_1) \Rightarrow s_3}$$

## Examples: derivation trees

$$\frac{\vdots}{("x'' ::= N\ 5;; "y'' ::= V\ "x'',\ s) \Rightarrow ?} \qquad \frac{\vdots}{(w,\ s_i) \Rightarrow ?}$$

where

$$\begin{aligned} w &= \textit{WHILE}\ b\ \textit{DO}\ c \\ b &= \textit{NotEq}\ (V\ "x'')\ (N\ 2) \\ c &= "x'' ::= \textit{Plus}\ (V\ "x'')\ (N\ 1) \\ s_i &= s("x'' := i) \end{aligned}$$

$$\begin{aligned} \textit{NotEq}\ a_1\ a_2 &= \\ \textit{Not}(\textit{And}\ (&\textit{Not}(\textit{Less}\ a_1\ a_2))\ (\textit{Not}(\textit{Less}\ a_2\ a_1)))) \end{aligned}$$

Logically speaking

$$(c, s) \Rightarrow t$$

is just infix syntax for

$$big\_step\ (c,s)\ t$$

where

$$big\_step :: com \times state \Rightarrow state \Rightarrow bool$$

is an inductively defined predicate.



# Big\_Step.thy

Semantics





## Rule inversion

What can we deduce from

- $(SKIP, s) \Rightarrow t$  ?
- $(x ::= a, s) \Rightarrow t$  ?
- $(c_1;; c_2, s_1) \Rightarrow s_3$  ?
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \Rightarrow t$  ?
- $(w, s) \Rightarrow t$  where  $w = WHILE\ b\ DO\ c$  ?

# Automating rule inversion

Isabelle command **inductive\_cases** produces theorems that perform rule inversions automatically.



We reformulate the inverted rules. Example:

$$\frac{(c_1;; c_2, s_1) \Rightarrow s_3}{\exists s_2. (c_1, s_1) \Rightarrow s_2 \wedge (c_2, s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{\bigwedge s_2. [(c_1, s_1) \Rightarrow s_2; (c_2, s_2) \Rightarrow s_3] \implies P}{P}$$



Replaces assem  $(c_1;; c_2, s_1) \Rightarrow s_3$  by two assems  $(c_1, s_1) \Rightarrow s_2$  and  $(c_2, s_2) \Rightarrow s_3$  (with a new fixed  $s_2$ ).

No  $\exists$  and  $\wedge$ !

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \dots \quad asm_n \Longrightarrow P}{P}$$

(possibly with  $\bigwedge \bar{x}$  in front of the  $asm_i \Longrightarrow P$ )

Reading:

To prove a goal  $P$  with assumption  $asm$ ,  
prove all  $asm_i \Longrightarrow P$

Example:

$$\frac{F \vee G \quad F \Longrightarrow P \quad G \Longrightarrow P}{P}$$

## *elim* attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*
- Can also be added locally, eg (*blast elim: ...*)
- Variant: *elim!* applies elim-rules eagerly.

# Big\_Step.thy

Rule inversion



# Command equivalence

Two commands have the same input/output behaviour:

$$c \sim c' \equiv (\forall s\ t. (c, s) \Rightarrow t \longleftrightarrow (c', s) \Rightarrow t)$$



## Example

$$w \sim w'$$



where  $w = \text{WHILE } b \text{ DO } c$

$w' = \text{IF } b \text{ THEN } c;; w \text{ ELSE SKIP}$



# Equivalence proof

$$\begin{aligned} & (w, s) \Rightarrow t \\ & \longleftrightarrow \\ & bval\ b\ s \wedge (\exists s'. (c, s) \Rightarrow s' \wedge (w, s') \Rightarrow t) \\ & \quad \vee \\ & \neg\ bval\ b\ s \wedge t = s \\ & \longleftrightarrow \\ & (w', s) \Rightarrow t \end{aligned}$$

Using the rules and rule inversions for  $\Rightarrow$ .

# Big\_Step.thy

Command equivalence

# Execution is deterministic

Any two executions of the same command in the same start state lead to the same final state:

$$(c, s) \Rightarrow t \implies (c, s) \Rightarrow t' \implies t = t'$$

Proof by rule induction, for arbitrary  $t'$ .



# Big\_Step.thy

Execution is deterministic

# The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

$(c, s)$  does not terminate iff  $\nexists t. (c, s) \Rightarrow t$ ?

Needs a formal notion of nontermination to prove it.  
Could be wrong if we have forgotten a  $\Rightarrow$  rule.

Big-step semantics cannot directly describe

- nonterminating computations,
- parallel computations.

We need a finer grained semantics!

① IMP Commands

② Big-Step Semantics

③ Small-Step Semantics

# Small-step semantics

Concrete syntax:

$$(com, state) \rightarrow (com, state)$$

Intended meaning of  $(c, s) \rightarrow (c', s')$ :

*The first step in the execution of  $c$  in state  $s$  leaves a “remainder” command  $c'$  to be executed in state  $s'$ .*

Execution as finite or infinite reduction:

$$(c_1, s_1) \rightarrow (c_2, s_2) \rightarrow (c_3, s_3) \rightarrow \dots$$



# Terminology

- A pair  $(c,s)$  is called a *configuration*.
- If  $cs \rightarrow cs'$  we say that  $cs$  *reduces* to  $cs'$ .
- A configuration  $cs$  is *final* iff  $\nexists cs'. cs \rightarrow cs'$

The intention:

$(SKIP, s)$  is final

Why?

*SKIP* is the empty program. Nothing more to be done.

# Small-step rules

$$(x ::= a, s) \rightarrow (SKIP, s(x := \text{aval } a \ s))$$

$$(SKIP;; c, s) \rightarrow (c, s)$$

$$\frac{(c_1, s) \rightarrow (c'_1, s')}{(c_1;; c_2, s) \rightarrow (c'_1;; c_2, s')}$$

## Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \rightarrow (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \rightarrow (c_2, s)}$$

$$(WHILE\ b\ DO\ c, s) \rightarrow (IF\ b\ THEN\ c;;\ WHILE\ b\ DO\ c\ ELSE\ SKIP, s)$$

**Fact**  $(SKIP, s)$  is a final configuration.

## Small-step examples

$$("z'' ::= V "x'';; "x'' ::= V "y'';; "y'' ::= V "z'', s) \rightarrow \dots$$

where  $s = \langle "x'' := 3, "y'' := 7, "z'' := 5 \rangle$ .

$$(w, s_0) \rightarrow \dots$$

where

$$\begin{aligned} w &= \text{WHILE } b \text{ DO } c \\ b &= \text{Less } (V "x'') (N 1) \\ c &= "x'' ::= \text{Plus } (V "x'') (N 1) \\ s_n &= \langle "x'' := n \rangle \end{aligned}$$

# Small\_Step.thy

Semantics

Are big and small-step semantics equivalent?

## From $\Rightarrow$ to $\rightarrow^*$

**Theorem**  $cs \Rightarrow t \implies cs \rightarrow^* (SKIP, t)$

Proof by rule induction (of course on  $cs \Rightarrow t$ )

In two cases a lemma is needed:

**Lemma**

$(c_1, s) \rightarrow^* (c_1', s') \implies (c_1;; c_2, s) \rightarrow^* (c_1';; c_2, s')$

Proof by rule induction.



## From $\rightarrow^*$ to $\Rightarrow$

**Theorem**  $cs \rightarrow^* (SKIP, t) \implies cs \Rightarrow t$

Proof by rule induction on  $cs \rightarrow^* (SKIP, t)$ .

In the induction step a lemma is needed:

**Lemma**  $cs \rightarrow cs' \implies cs' \Rightarrow t \implies cs \Rightarrow t$

Proof by rule induction on  $cs \rightarrow cs'$ .

# Equivalence

**Corollary**  $cs \Rightarrow t \iff cs \rightarrow^* (SKIP, t)$

# Small\_Step.thy

Equivalence of big and small

## Can execution stop prematurely?

That is, are there any final configs except  $(SKIP, s)$  ?

**Lemma**  $final(c, s) \implies c = SKIP$

We prove the contrapositive

$$c \neq SKIP \implies \neg final(c, s)$$

by induction on  $c$ .

- Case  $c_1;; c_2$ : by case distinction:
  - $c_1 = SKIP \implies \neg final(c_1;; c_2, s)$
  - $c_1 \neq SKIP \implies \neg final(c_1, s)$  (by IH)  
 $\implies \neg final(c_1;; c_2, s)$
- Remaining cases: trivial or easy

By rule inversion:  $(SKIP, s) \rightarrow ct \implies False$

Together:

**Corollary**  $final(c, s) = (c = SKIP)$

# Infinite executions

$\Rightarrow$  yields final state iff  $\rightarrow$  terminates

**Lemma**  $(\exists t. cs \Rightarrow t) = (\exists cs'. cs \rightarrow^* cs' \wedge \text{final } cs')$

Proof:  $(\exists t. cs \Rightarrow t)$   
=  $(\exists t. cs \rightarrow^* (\text{SKIP}, t))$   
    (by big = small)  
=  $(\exists cs'. cs \rightarrow^* cs' \wedge \text{final } cs')$   
    (by final = SKIP)

Equivalent:

$\Rightarrow$  does not yield final state iff  $\rightarrow$  does not terminate

# May versus Must

$\rightarrow$  is deterministic:

**Lemma**  $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$   
(Proof by rule induction)

Therefore: no difference between

**may** terminate (there is a terminating  $\rightarrow$  path)

**must** terminate (all  $\rightarrow$  paths terminate)

Therefore:  $\Rightarrow$  correctly reflects termination behaviour.

With nondeterminism: may have both  $cs \Rightarrow t$  and a nonterminating reduction  $cs \rightarrow cs' \rightarrow \dots$

# Chapter 8

## Hoare Logic



④ Partial Correctness

⑤ Verification Conditions

⑥ Total Correctness

④ Partial Correctness

⑤ Verification Conditions

⑥ Total Correctness

## ④ Partial Correctness

Introduction

The Syntactic Approach

The Semantic Approach

Soundness and Completeness

We have proved functional programs correct  
(e.g. a compiler).

We have proved properties of imperative languages  
(e.g. type safety).

But how do we prove properties of imperative programs?

An example program:

*"y" ::= N 0;; wsum*

where

*wsum*  $\equiv$

*WHILE Less (N 0) (V "x")*  
*DO ("y" ::= Plus (V "y") (V "x"));*  
*"x" ::= Plus (V "x") (N (- 1)))*

At the end of the execution of *"y" ::= N 0;; wsum* variable *"y"* should contain the sum  $1 + \dots + i$  where *i* is the initial value of *"x"*.

*sum i = (if  $i \leq 0$  then 0 else sum (i - 1) + i)*

# A proof via operational semantics

Theorem:

$$("y'' ::= N\ 0;;\ wsum,\ s) \Rightarrow t \Longrightarrow \\ t\ "y'' = sum\ (s\ "x'')$$

Required Lemma:

$$(wsum,\ s) \Rightarrow t \Longrightarrow \\ t\ "y'' = s\ "y'' + sum\ (s\ "x'')$$

Proved by rule induction.

*Hoare Logic* provides a *structured* approach for reasoning about properties of states during program execution:

- Rules of Hoare Logic (almost) syntax directed
- Automates reasoning about program execution
- No explicit induction

But no free lunch:

- Must prove implications between predicates on states
- Needs *invariants*.

## ④ Partial Correctness

Introduction

**The Syntactic Approach**

The Semantic Approach

Soundness and Completeness



This is the standard approach.

Formulas are syntactic objects.

Everything is very concrete and simple.

But complex to formalize.

Hence we soon move to a semantic view of formulas.

Reason for introduction of syntactic approach: didactic

For now, we work with a (syntactically) simplified version of IMP.

Hoare Logic reasons about *Hoare triples*  $\{P\} \ c \ \{Q\}$  where

- $P$  and  $Q$  are *syntactic formulas* involving program variables
- $P$  is the *precondition*,  $Q$  is the *postcondition*
- $\{P\} \ c \ \{Q\}$  means that  
if  $P$  is true at the start of the execution,  
 $Q$  is true at the end of the execution  
— if the execution terminates! (*partial correctness*)

Informal example:

$$\{x = 41\} \ x := x + 1 \ \{x = 42\}$$

Terminology:  $P$  and  $Q$  are called *assertions*.

# Examples

$$\{x = 5\} \quad ? \quad \{x = 10\}$$

$$\{True\} \quad ? \quad \{x = 10\}$$

$$\{x = y\} \quad ? \quad \{x \neq y\}$$

Boundary cases:

$$\{True\} \quad ? \quad \{True\}$$

$$\{True\} \quad ? \quad \{False\}$$

$$\{False\} \quad ? \quad \{Q\}$$

# The rules of Hoare Logic

$$\{P\} \text{ SKIP } \{P\}$$

$$\{Q[a/x]\} x := a \{Q\}$$

Notation:  $Q[a/x]$  means “ $Q$  with  $a$  substituted for  $x$ ”.

Examples:

$\{$	$\}$	$x := 5$	$\{x = 5\}$
$\{$	$\}$	$x := x+5$	$\{x = 5\}$
$\{$	$\}$	$x := 2*(x+5)$	$\{x > 20\}$

Alternative explanation of assignment rule:

$$\{Q[a]\} x := a \{Q[x]\}$$

The assignment axiom allows us  
to compute the precondition from the postcondition.

There is a version to compute the postcondition from  
the precondition, but it is more complicated. (Exercise!)

## More rules of Hoare Logic

$$\frac{\{P_1\} \ c_1 \ \{P_2\} \quad \{P_2\} \ c_2 \ \{P_3\}}{\{P_1\} \ c_1; c_2 \ \{P_3\}}$$

$$\frac{\{P \wedge b\} \ c_1 \ \{Q\} \quad \{P \wedge \neg b\} \ c_2 \ \{Q\}}{\{P\} \ \text{IF } b \ \text{THEN } c_1 \ \text{ELSE } c_2 \ \{Q\}}$$

$$\frac{\{P \wedge b\} \ c \ \{P\}}{\{P\} \ \text{WHILE } b \ \text{DO } c \ \{P \wedge \neg b\}}$$

In the While-rule,  $P$  is called an *invariant* because it is preserved across executions of the loop body.

# The *consequence* rule

So far, the rules were syntax-directed. Now we add

$$\frac{P' \longrightarrow P \quad \{P\} \text{ c } \{Q\} \quad Q \longrightarrow Q'}{\{P'\} \text{ c } \{Q'\}}$$

*Preconditions can be strengthened,  
postconditions can be weakened.*

## Two derived rules

Problem with assignment and While-rule:  
special form of pre and postcondition.  
Better: combine with consequence rule.

$$\frac{P \longrightarrow Q[a/x]}{\{P\} \ x := a \ \{Q\}}$$

$$\frac{\{P \wedge b\} \ c \ \{P\} \quad P \wedge \neg b \longrightarrow Q}{\{P\} \ \text{WHILE } b \ \text{DO } c \ \{Q\}}$$



# Example

$\{x = i\}$

$y := 0;$

*WHILE*  $0 < x$  *DO* ( $y := y+x; x := x-1$ )

$\{y = \text{sum } i\}$

Example proof exhibits key properties of Hoare logic:

- Choice of rules is syntax-directed and hence automatic.
- Proof of “;” proceeds from right to left.
- Proofs require only invariants and arithmetic reasoning.

## ④ Partial Correctness

Introduction

The Syntactic Approach

**The Semantic Approach**

Soundness and Completeness

Assertions are predicates on states

$$assn = state \Rightarrow bool$$

Alternative view: *sets of states*

Semantic approach simplifies meta-theory, our main objective.

# Validity

$$\models \{P\} \text{ c } \{Q\}$$

$$\longleftrightarrow$$

$$\forall s \ t. \ P \ s \wedge (c, s) \Rightarrow t \longrightarrow Q \ t$$

“ $\{P\} \text{ c } \{Q\}$  is valid”

In contrast:

$$\vdash \{P\} \text{ c } \{Q\}$$

“ $\{P\} \text{ c } \{Q\}$  is provable/derivable”

# Provability

$$\vdash \{P\} \text{ SKIP } \{P\}$$

$$\vdash \{\lambda s. Q (s[a/x])\} x ::= a \{Q\}$$

$$\text{where } s[a/x] \equiv s(x := \text{aval } a \text{ } s)$$

Example:  $\{x+5 = 5\} x := x+5 \{x = 5\}$  in semantic terms:

$$\vdash \{P\} x ::= \text{Plus } (V \ x) \ (N \ 5) \ \{\lambda t. t \ x = 5\}$$

$$\begin{aligned} \text{where } P &= (\lambda s. (\lambda t. t \ x = 5)(s[\text{Plus } (V \ x) \ (N \ 5)/x])) \\ &= (\lambda s. (\lambda t. t \ x = 5)(s(x := s \ x + 5))) \\ &= (\lambda s. s \ x + 5 = 5) \end{aligned}$$

$$\frac{\vdash \{P\} \ c_1 \ \{Q\} \quad \vdash \{Q\} \ c_2 \ \{R\}}{\vdash \{P\} \ c_1;; \ c_2 \ \{R\}}$$

$$\frac{\begin{array}{l} \vdash \{\lambda s. P \ s \wedge \text{bval } b \ s\} \ c_1 \ \{Q\} \\ \vdash \{\lambda s. P \ s \wedge \neg \text{bval } b \ s\} \ c_2 \ \{Q\} \end{array}}{\vdash \{P\} \ \text{IF } b \ \text{THEN } c_1 \ \text{ELSE } c_2 \ \{Q\}}$$

$$\frac{\vdash \{\lambda s. P \ s \wedge \text{bval } b \ s\} \ c \ \{P\}}{\vdash \{P\} \ \text{WHILE } b \ \text{DO } c \ \{\lambda s. P \ s \wedge \neg \text{bval } b \ s\}}$$

$$\frac{
\begin{array}{c}
\forall s. P' s \longrightarrow P s \\
\vdash \{P\} c \{Q\} \\
\forall s. Q s \longrightarrow Q' s
\end{array}
}{
\vdash \{P'\} c \{Q'\}
}$$



Hoare\_Examples.thy

## ④ Partial Correctness

Introduction

The Syntactic Approach

The Semantic Approach

Soundness and Completeness

# Soundness

Everything that is provable is valid:

$$\vdash \{P\} \ c \ \{Q\} \implies \models \{P\} \ c \ \{Q\}$$

Proof by induction, with a nested induction in the While-case.

Towards completeness:  $\models \Rightarrow \vdash$

# Weakest preconditions

The **weakest precondition**  
of command  $c$  w.r.t. postcondition  $Q$ :

$$wp\ c\ Q = (\lambda s. \forall t. (c, s) \Rightarrow t \longrightarrow Q\ t)$$

The set of states that lead (via  $c$ ) into  $Q$ .

A foundational semantic notion, not merely for the completeness proof.

## Nice and easy properties of $wp$

$$wp \text{ SKIP } Q = Q$$

$$wp (x ::= a) Q = (\lambda s. Q (s[a/x]))$$

$$wp (c_1;; c_2) Q = wp \ c_1 \ (wp \ c_2 \ Q)$$

$$wp (IF \ b \ THEN \ c_1 \ ELSE \ c_2) Q = \\ (\lambda s. \text{if } bval \ b \ s \text{ then } wp \ c_1 \ Q \ s \text{ else } wp \ c_2 \ Q \ s)$$

$$\neg bval \ b \ s \implies wp \ (WHILE \ b \ DO \ c) \ Q \ s = Q \ s$$

$$bval \ b \ s \implies \\ wp \ (WHILE \ b \ DO \ c) \ Q \ s = \\ wp \ (c;; WHILE \ b \ DO \ c) \ Q \ s$$

# Completeness

$$\models \{P\} \ c \ \{Q\} \implies \vdash \{P\} \ c \ \{Q\}$$

Proof idea: do not prove  $\vdash \{P\} \ c \ \{Q\}$  directly,  
prove something stronger:

**Lemma**  $\vdash \{wp \ c \ Q\} \ c \ \{Q\}$

**Proof** by induction on  $c$ , for arbitrary  $Q$ .

Now prove  $\vdash \{P\} \ c \ \{Q\}$  from  $\vdash \{wp \ c \ Q\} \ c \ \{Q\}$   
by the consequence rule because

**Fact**  $\models \{P\} \ c \ \{Q\} \iff (\forall s. P \ s \longrightarrow wp \ c \ Q \ s)$

Follows directly from defs of  $\models$  and  $wp$ .

$$\vdash \{P\} \ c \ \{Q\} \iff \models \{P\} \ c \ \{Q\}$$

Proving program properties by Hoare logic ( $\vdash$ )  
is just as powerful as by operational semantics ( $\models$ ).



## WARNING

Most texts that discuss completeness of Hoare logic state or prove that Hoare logic is only “relatively complete” but **not complete**.

Reason: the standard notion of completeness assumes some abstract mathematical notion of  $\models$ .

Our notion of  $\models$  is defined within the same (limited) proof system (for HOL) as  $\vdash$ .

④ Partial Correctness

⑤ Verification Conditions

⑥ Total Correctness

Idea:

*Reduce provability in Hoare logic to provability in the assertion language:  
automate the Hoare logic part of the problem.*

More precisely:

*From  $\{P\} \ c \ \{Q\}$  generate an assertion  $A$ ,  
the **verification condition**,  
such that  $\vdash \{P\} \ c \ \{Q\}$  iff  $A$  is provable.*

Method:

*Simulate syntax-directed application of Hoare logic rules. Collect all assertion language side conditions.*

# A problem: loop invariants

Where do they come from?

A trivial solution:

Let the user provide them!

How?

Each loop must be annotated with its invariant!

How to synthesize loop invariants automatically  
is an important research problem.

Which we ignore for the moment.

But come back to later.

Terminology:

VCG = Verification Condition Generator

All successful verification technology for imperative programs relies on

- VCGs (of one kind or another)
- and powerful (semi-)automatic theorem provers.

# The (approx.) plan of attack

- 1 Introduce **annotated** commands with loop invariants
- 2 Define functions for *computing*
  - weakest preconditions:  $pre :: com \Rightarrow assn \Rightarrow assn$
  - verification conditions:  $vc :: com \Rightarrow assn \Rightarrow bool$
- 3 Soundness:  $vc\ c\ Q \implies \vdash \{ ? \}\ c\ \{ Q \}$
- 4 Completeness: if  $\vdash \{ P \}\ c\ \{ Q \}$  then  $c$  can be annotated (becoming  $C$ ) such that  $vc\ C\ Q$ .

The details are a bit different ...

# Annotated commands

Like commands, except for *While*:

**datatype** *acom* = *Askip*  
                  | *Aassign vname aexp*  
                  | *Aseq acom acom*  
                  | *Aif bexp acom acom*  
                  | *Awhile assn bexp acom*

Concrete syntax: like commands, except for *WHILE*:

*{I} WHILE b DO c*



# Weakest precondition

$$pre :: acom \Rightarrow assn \Rightarrow assn$$

$$pre \text{ SKIP } Q = Q$$

$$pre (x ::= a) Q = (\lambda s. Q (s[a/x]))$$

$$pre (C_1;; C_2) Q = pre C_1 (pre C_2 Q)$$

$$pre (IF b THEN C_1 ELSE C_2) Q = \\ (\lambda s. \text{if } bval\ b\ s \text{ then } pre\ C_1\ Q\ s \text{ else } pre\ C_2\ Q\ s)$$

$$pre (\{I\} \text{ WHILE } b \text{ DO } C) Q = I$$

## Warning

In the presence of loops,  
*pre C* may not be the weakest precondition  
but may be anything!

# Verification condition

$vc :: acom \Rightarrow assn \Rightarrow bool$

$vc \text{ SKIP } Q = True$

$vc (x ::= a) Q = True$

$vc (C_1;; C_2) Q = (vc C_1 (pre C_2 Q) \wedge vc C_2 Q)$

$vc (IF b THEN C_1 ELSE C_2) Q =$   
 $(vc C_1 Q \wedge vc C_2 Q)$

$vc (\{I\} WHILE b DO C) Q =$   
 $((\forall s. (I s \wedge bval b s \longrightarrow pre C I s) \wedge$   
 $(I s \wedge \neg bval b s \longrightarrow Q s)) \wedge$   
 $vc C I)$

Verification conditions only arise from loops:

- the invariant must be invariant
- and it must imply the postcondition.

Everything else in the definition of *vc* is just bureaucracy: collecting assertions and passing them around.

Hoare triples operate on *com*,  
functions *pre* and *vc* operate on *acom*.  
Therefore we define

$$\textit{strip} :: \textit{acom} \Rightarrow \textit{com}$$

$$\textit{strip} \textit{SKIP} = \textit{SKIP}$$

$$\textit{strip} (x ::= a) = x ::= a$$

$$\textit{strip} (C_1;; C_2) = \textit{strip} C_1;; \textit{strip} C_2$$

$$\textit{strip} (\textit{IF } b \textit{ THEN } C_1 \textit{ ELSE } C_2) = \\ \textit{IF } b \textit{ THEN } \textit{strip} C_1 \textit{ ELSE } \textit{strip} C_2$$

$$\textit{strip} (\{I\} \textit{ WHILE } b \textit{ DO } C) = \textit{WHILE } b \textit{ DO } \textit{strip} C$$

# Soundness of *vc* & *pre* w.r.t. $\vdash$

$$vc\ C\ Q \implies \vdash \{pre\ C\ Q\}\ strip\ C\ \{Q\}$$

Proof by induction on  $C$ , for arbitrary  $Q$ .

Corollary:

$$\llbracket vc\ C\ Q; \forall s. P\ s \longrightarrow pre\ C\ Q\ s \rrbracket \\ \implies \vdash \{P\}\ strip\ C\ \{Q\}$$

How to prove some  $\vdash \{P\}\ c\ \{Q\}$ :

- Annotate  $c$  yielding  $C$ , i.e.  $strip\ C = c$ .
- Prove Hoare-free premise of corollary.

But is premise provable if  $\vdash \{P\}\ c\ \{Q\}$  is?

$$\llbracket vc \ C \ Q; \forall s. \ P \ s \longrightarrow pre \ C \ Q \ s \rrbracket \\ \implies \vdash \{P\} \ strip \ C \ \{Q\}$$

Why could premise not be provable although conclusion is?

- Some annotation in  $C$  is not invariant.
- $vc$  or  $pre$  are wrong  
(e.g. accidentally always produce *False*).

Therefore we prove completeness:  
suitable annotations exist such that premise is provable.

# Completeness of *vc* & *pre* w.r.t. $\vdash$

$$\vdash \{P\} \ c \ \{Q\} \implies \\ \exists C. \text{strip } C = c \wedge vc \ C \ Q \wedge (\forall s. P \ s \longrightarrow pre \ C \ Q \ s)$$

Proof by rule induction. Needs two monotonicity lemmas:

$$\llbracket \forall s. P \ s \longrightarrow P' \ s; pre \ C \ P \ s \rrbracket \implies pre \ C \ P' \ s$$

$$\llbracket \forall s. P \ s \longrightarrow P' \ s; vc \ C \ P \rrbracket \implies vc \ C \ P'$$



④ Partial Correctness

⑤ Verification Conditions

⑥ Total Correctness

- Partial Correctness:  
*if* command terminates, postcondition holds
- Total Correctness:  
command terminates *and* postcondition holds

Total Correctness = Partial Correctness + Termination

Formally:

$$\begin{aligned} &(\models_t \{P\} c \{Q\}) = \\ &(\forall s. P s \longrightarrow (\exists t. (c, s) \Rightarrow t \wedge Q t)) \end{aligned}$$

Assumes that semantics is deterministic!

Exercise: Reformulate for nondeterministic language

# $\vdash_t$ : A proof system for total correctness

Only need to change the *WHILE* rule.

Some measure function  $state \Rightarrow nat$   
must decrease with every loop iteration

$$\frac{\bigwedge n. \vdash_t \{ \lambda s. P\ s \wedge bval\ b\ s \wedge n = f\ s \}\ c\ \{ \lambda s. P\ s \wedge f\ s < n \}}{\vdash_t \{ P \}\ WHILE\ b\ DO\ c\ \{ \lambda s. P\ s \wedge \neg\ bval\ b\ s \}}$$

*WHILE* rule can be generalized from a function to a relation:

$$\frac{\bigwedge n. \vdash_t \{ \lambda s. P\ s \wedge \text{bval}\ b\ s \wedge T\ s\ n \} \ c \ \{ \lambda s. P\ s \wedge (\exists n' < n. T\ s\ n') \}}{\vdash_t \{ \lambda s. P\ s \wedge (\exists n. T\ s\ n) \} \ \text{WHILE}\ b\ \text{DO}\ c \ \{ \lambda s. P\ s \wedge \neg \text{bval}\ b\ s \}}$$

# Hoare\_Total.thy

Example

# Soundness

$$\vdash_t \{P\} c \{Q\} \implies \models_t \{P\} c \{Q\}$$

Proof by induction, with a nested induction on  $n$  in the While-case.

# Completeness

$$\models_t \{P\} \ c \ \{Q\} \Longrightarrow \vdash_t \{P\} \ c \ \{Q\}$$

Follows easily from

$$\vdash_t \{wp_t \ c \ Q\} \ c \ \{Q\}$$

where

$$wp_t \ c \ Q = (\lambda s. \exists t. (c, s) \Rightarrow t \wedge Q \ t).$$

Proof of  $\vdash_t \{wp_t \ c \ Q\} \ c \ \{Q\}$  is by induction on  $c$ .

In the *WHILE*  $b \ DO \ c$  case, use the *WHILE* rule with

$$\frac{\neg \text{bval } b \ s}{T \ s \ 0} \qquad \frac{\text{bval } b \ s \quad (c, s) \Rightarrow s' \quad T \ s' \ n}{T \ s \ (n + 1)}$$

$T \ s \ n$  means that *WHILE*  $b \ DO \ c$  started in state  $s$  needs  $n$  iterations to terminate.