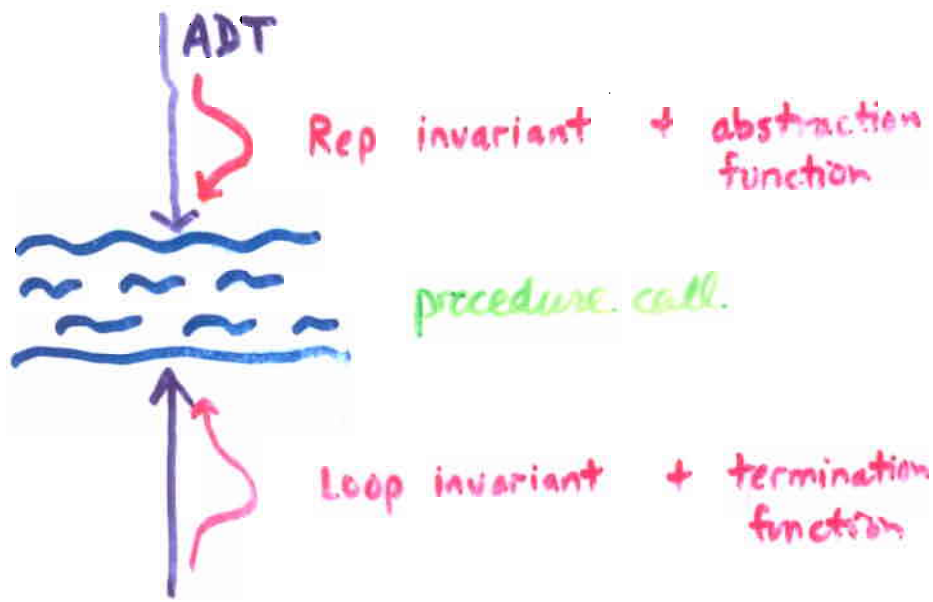
Specification Aids for Program Development



Axiomatic Method (Hoare)

Notation

$$\{P\}\ Q\ \{R\}$$

program

specification

Meaning

Partial correctness interpretation:

If the assertion P is true before initiation of program Q
→ and if execution terminates,
then the assertion R will be true upon termination.

Total correctness interpretation:

If the assertion P is true before initiation of program Q
then Q will terminate
and the assertion R will be true upon termination.

Hoare Logic

Assignment axiom

$$P^{e}_{x} \quad \{x := e\} \quad P$$

Composition rule

$$\frac{P\ \{Q_1\}\ R_1 \ ,\ R_1\ \{Q_2\}\ R}{P\ \{Q_1 ; Q_2\}\ R}$$

Iteration rule

$$\frac{P \wedge B\ \{S\}\ P}{P\ \{\underline{\text{while}}\ B\ \underline{\text{do}}\ S\}\ \neg B \wedge P}$$

Conditional rule

$$\frac{P \wedge B\ \{S_1\}\ Q \ ,\ P \wedge \neg B\ \{S_2\}\ Q}{P\ \{\underline{\text{if}}\ B\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\}\ Q}$$

Rules of consequence

$$\frac{P\ \{Q\}\ R \ ,\ S \Rightarrow P}{S\ \{Q\}\ R} \qquad \frac{P\ \{Q\}\ R \quad R \Rightarrow S}{P\ \{Q\}\ S}$$

Examples

$\{x \geq 0 \ \& \ y \geq 1\}$

```
q := 0;
r := x;
while r ≥ y do
    begin q := q + 1;
          r := r - y
    end
```

$\{q = x/y \ \& \ r = x \bmod y\}$

division $\frac{x}{y}$

$\{y \geq 0\}$

```
z := 1;
t := 1
while t ≤ y do
    begin z := z * x;
          t := t + 1
    end
```

$\{z = x ** y\}$

exponentiation x^y

Weakest Pre-condition (wp)

Def wp(S, R) = set of all possible states that S will map into states of R.

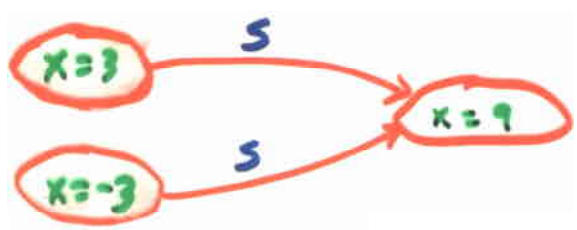
N.B. "predicate" = "set of states"

purpose: defines what start states will get you to desired final state(s).

N.B. many-to-1

e.g., S: x := x ** 2

R: x = 9



so, wp(S, R) = x=3 ∨ x=-3

Example and Two Questions

Consider $y := x/k$

$\{k \neq 0\}\ y := x/k\ \{x = yk\}$ True

could have a weaker spec:

$\{k = 1\}\ y := x/k\ \{x = y\}$ True

or

$\{x > 0 \wedge k > 0\}\ y := x/k\ \{y > 0\}$ True

False assertion:

$\{k = 1\}\ y := x/k\ \{y > x\}$ False

Two Q's:

1. How does one specify Q and R ?
2. How does one prove assertion $\{Q\}\ S\ \{R\}$?

Answers:

1. Art
2. Use wp recursively defined over statements in S.

Wp for Simple Statements

1. null statement

   $wp(\underline{skip}, R) = R$

2. halt

   $wp(\underline{abort}, R) = false$

3. sequential composition

   $wp(S1; S2, R) = wp(S1, wp(S2, R))$

4. assignment

   $wp(x := e, R) = domain(E)\ \underline{cand}\ R^x_e$

5. conditional

$$
\begin{aligned}
wp(\ & \underline{if}\ B_1 \rightarrow S_1 \\
& \square\ B_2 \rightarrow S_2 \\
& \vdots \\
& \square\ B_n \rightarrow S_n\ \underline{fi},\ R) = domain(BB) \wedge BB \wedge \\
& \qquad B_1 \Rightarrow wp(S_1, R) \wedge \\
& \qquad \vdots \qquad\qquad \vdots \\
& \qquad B_n \Rightarrow wp(S_n, R)
\end{aligned}
$$

where $BB = B_1 \vee \cdots \vee B_n$

Proof Technique

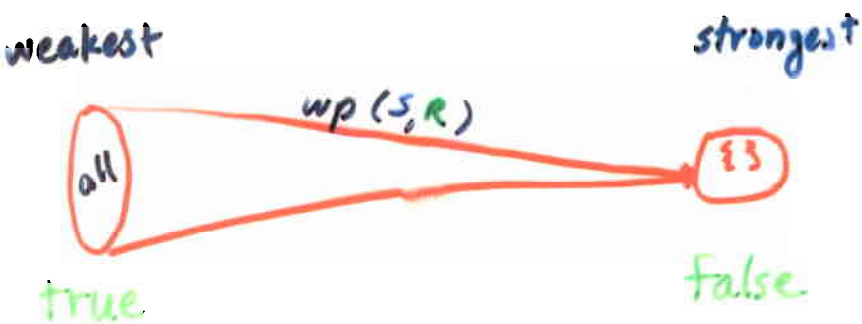
Q2)

Given $\{Q\}\ S\ \{R\}$

1. Find $wp(S, R)$ (mechanical)

2. Prove $Q \Rightarrow wp(S, R)$ (just a predicate)

N.B. Q might be stronger or equivalent:



Example

$\{x \neq 0\}$ <u>if</u> $x < 0 \rightarrow x := -x \;\square\; x \geq 0 \rightarrow x := x - 1$ <u>fi</u> $\{x \geq 0\}$

1. $wp(\underline{\text{if}}\ x < 0 \rightarrow x := -x \;\square\; x \geq 0 \rightarrow x := x - 1\ \underline{\text{fi}},\ x \geq 0)$

$$= (x < 0 \vee x \geq 0) \wedge x < 0 \Rightarrow wp(x := -x, x \geq 0)$$
$$\wedge\ x \geq 0 \Rightarrow wp(x := x - 1, x \geq 0)$$
$$= T \wedge x < 0 \Rightarrow -x \geq 0 \wedge x \geq 0 \Rightarrow x - 1 \geq 0$$

2. $Q \Rightarrow wp$

$$x \neq 0 \Rightarrow (x < 0 \Rightarrow -x \geq 0 \wedge x \geq 0 \Rightarrow x \geq 0)$$

Loops

S: $\quad DO = \underline{\text{do}} \quad B1 \rightarrow S1$

$\quad\quad\quad\quad \square \quad \vdots$

$\quad\quad\quad\quad \square \; B_n \rightarrow S_n$

$\quad\quad\quad \underline{\text{od}}$

$wp(DO, R) = (\exists k : 0 \leq k : H_k(R))$

where $H_k(R)$ represents the set of all states in which execution of DO terminates in k of fewer iterations with R true.

$H_0(R) = \neg BB \wedge R \quad , k=0$

$H_k(R) = H_0(R) \vee wp(IF, H_{k-1}(R)) \quad , k>0$

YUCK

Loops

Thm: If $(P \wedge BB) \Rightarrow wp(IF, P)$ — P is INVARIANT

$\wedge\ (P \wedge BB) \Rightarrow t > 0$ — t is an int. TERMINAL f'cn

$\wedge\ (P \wedge BB) \Rightarrow wp(t1 := t;\ IF,\ t < t1)$ — t decreases

then

$P \Rightarrow wp(Do,\ P \wedge \neg BB)$

So, to prove loop, find P, t s.t. Thm. holds.
Then we are guaranteed wp holds.

Loop Checklist

Show:

1. INIT: P is initially true

2. INV: P is invariant

   $\{P \wedge B_i\}\ S_i\ \{P\}$ for $1 \le i \le n$

3. GOAL: R holds on termination

   $P \wedge \neg BB \Rightarrow R$

4. BD: t is bounded from below while in loop

   $P \wedge BB \Rightarrow t > 0$

5. DEC: t decreases at each loop iteration

   $\{P \wedge B_i\}\ t1 := t\ ;\ S_i\ \{t < t1\}$

Hard part: Finding P + t.

"Easy" part: Doing checklist.

Example (Sketch)

Q: true

S: $i, s := 1, b[0]$;

<u>do</u> $i < 11 \rightarrow i, s := i+1, s + b[i]$ <u>od</u>

R: $s = \sum_{0 \le k < 11} b[k]$

P: $1 \le i \le 11 \ \wedge \ s = \sum_{0 \le k < i} b[k]$

t: $11 - i$

Checklist

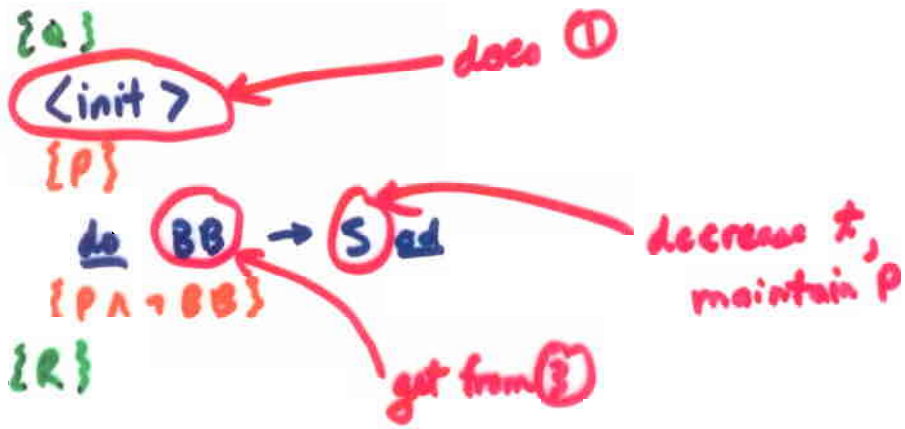
1. INIT: $\{true\}\ i, s := 1, b[0]\ \{P\}$
2. Show INV $\{P \wedge B\}\ i, s := i+1, s + b[i]\ \{P\}$

   2 parts: find wp, show $P \wedge B \Rightarrow$ wp.
3. Show GOAL $P \wedge \neg B \Rightarrow R$
4. Show BD $P \wedge B \Rightarrow (t > 0)$
5. Show DEC $\{P \wedge B\}\ t1 := t;\ S_i\ \{t < t1\}$

Developing Programs using R+P

1. Look at post-condition R. That is your goal.
   Derive program with goal in mind.

2. Loop:



Developing invariants:

a) Delete conjunct from R
b) Replace constant by variable.
c) Extend range of variable.

2. Thy's of invariants.

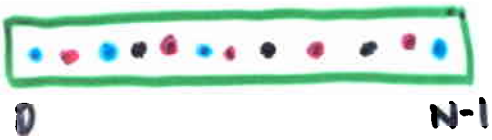
a) Garden Path

bound fkn t = distance
"Don't stray from path"

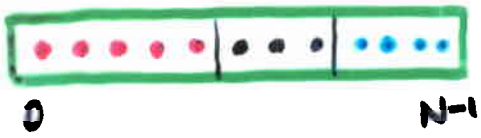
b) Balloon Thy

"Inflate set of final states to set of initial states."

Example: Dutch National Flag (Dijkstra)



R, W, B pebbles in buckets 0:N-1



Want pebbles in order of DNF

Operations

1. Fetch, F[i]

   Can see what's in only 1 bucket at a time

2. Swap, swap(F[i], F[j]) $i \neq j$

   Can swap the pebbles in 2 buckets i + j if $i \neq j$

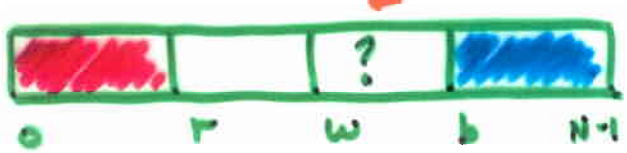
DNF (cont'd)

Notice: 1) RWB? or RW?WB or RW?B at any pt. in time
2) If we pick up a pebble we have to decide what to do in each case R, W, B.

Let's use RW?B as invariant:

P: F[0:r-1] red F[r:w-1] white F[w:b-1] ? F[b:N-1] blu

let's look at elements in here to decide what to do



A DNF solution

```
I: array F[0:N-1] of 3 colors
    r, w, b := 0, 0, N

    do  w ≠ b  →

        if  F[w] = red   →  swap (F[r], F[w]);
                            r, w := r+1, w+1

          [] F[w] = white →  w := w+1

          [] F[w] = blue  →  swap (F[b-1], F[w]);
                             b := b+1

        fi
    od

R:  F[0:r-1] red   F[r:w-1] white   F[w:N-1] blue
```

DNF (cont'd)

Checklist

1. INIT : all unknown so all other regions empty
   r, w, b := 0, 0, N

3. GUARD (GOAL): when unknown region becomes empty
   w = b

4. TERM (BD) b - w
   ⇒ decrease b or increase w

2. INV for each case

   if white ⇒ increase area for white ⇒ w := w + 1
   if red ⇒ shift white over by 1; increase area for red ⇒ swap; incr r, w
   if blue ⇒ increase area for blue + get new unknown ⇒ swap b-1, w; dec. b

5 DEC Show decrease in b - w

Procedure Call

Proc def'n

proc P (**value** $\bar{x}$; **value result** $\bar{y}$; **result** $\bar{z}$)

$\{P\}$ B $\{Q\}$

formals

pre-condition

post-condition

body

Proc call

$P(\bar{a}, \bar{b}, \bar{c})$

actuals

$wp(p(\bar{a}, \bar{b}, \bar{c}), R) = wp(\bar{x}, \bar{y} := \bar{a}, \bar{b};\ B;\ \bar{b}, \bar{c} := \bar{y}, \bar{z},\ R)$

This is fine formally, but what's wrong with this pragmatically?

- We want to make use of SPEC of proc. def'n w/o looking at IMPL (B).

Procedure Call (cont'd)

hm: Given proc def'n above. Then

$\{PR\}\ p(\bar{a},\bar{b},\bar{c})\ \{R\}$ holds. I.e., $PR \Rightarrow wp(p(\bar{a},\bar{b},\bar{c}), R)$

where

$$PR = P^{\bar{x},\bar{y}}_{\bar{a},\bar{b}} \wedge \left(\forall \bar{u},\bar{v} : Q^{\bar{y},\bar{z}}_{\bar{u},\bar{v}} \Rightarrow R^{\bar{b},\bar{c}}_{\bar{u},\bar{v}} \right)$$

says P must hold of the actuals

says if Q (post-condition) holds for any values of the formals upon termination, then it better imply that the desired post-condition R holds of the final values of the actuals

Perspective

Formal methods are useful for developing programs.

- at the module level, A + R
- at the statement level, P + *

There are many other formal methods for proving programs.
and giving program semantics.

- operational
- axiomatic — Hoare
- predicate — wp
- relational
- functional
- denotational