Refinement-Based Proofs of Functional Correctness

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

CSI Formal Methods Workshop
15 October 2014

Verifying Functional Correctness: Talk Outline

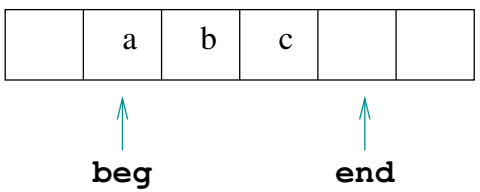
- What we mean by functional correctness
- Hoare logic based program verification
- Refinement theory
- Using VCC to prove refinement.

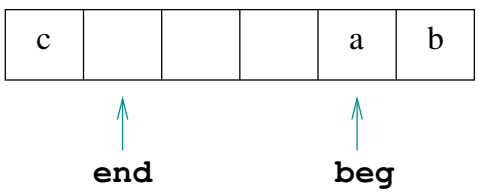
Outline of first talk

A C implementation of a queue

```
 1: int A[MAXLEN];
 2: unsigned beg, end, len;
 3:
 4: void init() {
 5:   beg = 0;
 6:   end = 0;
 7:   len = 0;
 8: }
 9:
10: int deq() { ...  }
```

```
11: void enq(int t) {
12:   if (len == MAXLEN)
13:     assert(0); /* exception */
14:   A[end] = t;
15:   if (end < MAXLEN-1)
16:     end++;
17:   else
18:     end = 0;
19:   len++;
20: }
```





Two ways to prove functional correctness...

First way: Directly annotate with pre and post conditions:

```
void enq(int newval)
_(requires \thread_local(A))
_(requires len < MAXLEN)
_(requires (beg < end) ==> (len == end - beg))
_(requires (beg == end) ==> (len == 0 || len == MAXLEN))
_(requires (beg > end) ==> (len == ((MAXLEN-beg) + end)))
_(ensures (\old(end) < MAXLEN-1) ==> ((end == \old(end) + 1) && (len == \old(le
_(ensures (\forall unsigned int n; (n != \old(end) ==> A[n] == \old(A[n]))))
_(ensures (A[\old(end)] == newval))
_(writes &len, &end, &A[end])
{
  A[end] = newval;
  if (end < MAXLEN-1)
    end++;
  else
    end = 0;
  len++;
}
```

Two ways to prove functional correctness...

Second way: Prove that implementation refines a simple abstract spec:

Abstract Spec

content: seq $\mathbb{Z}$

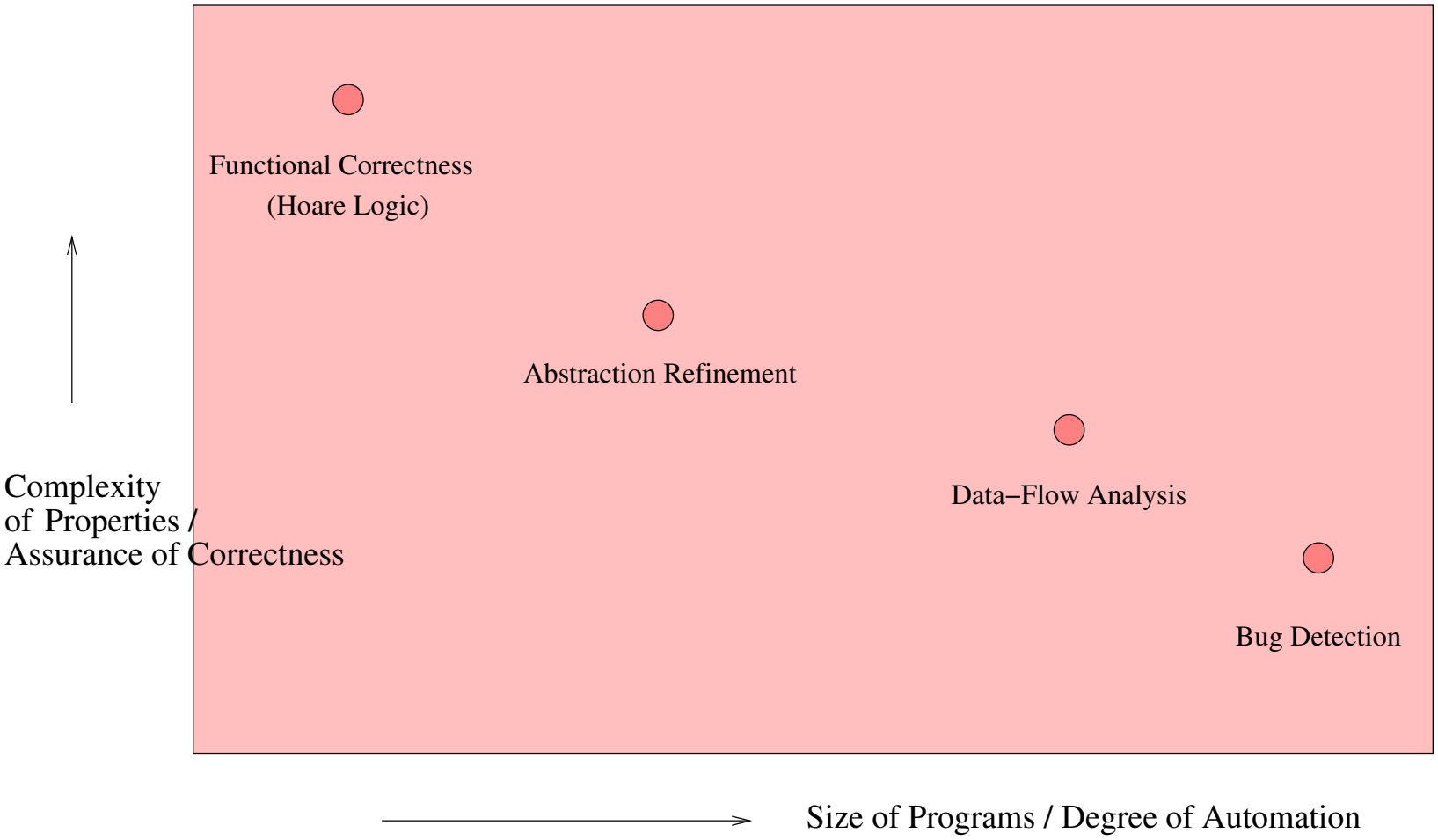
init():
  content' $= \langle\rangle$

enq(x: $\mathbb{Z}$):
  $\#content < k$
  content' $=$ *content* $\frown \langle x \rangle$

```
1: int A[MAXLEN];
2: unsigned beg, end, len;
3:
4: void init() {
5:   beg = 0;
6:   end = 0;
7:   len = 0;
8: }
9:
```

```
11: void enq(int t) {
12:   if (len == MAXLEN)
13:     assert(0); /* exception */
14:   A[end] = t;
15:   if (end < MAXLEN-1)
16:     end++;
17:   else
18:     end = 0;
19:   len++;
```

Different program verification techniques: Assurance vs Automation



Hoare Logic based verification can prove complex properties of programs like functional correctness and thus gives us highest-level of assurance about correctness of our software.

Program Verification

- Unlike testing, can prove that a program behaves correctly for all its inputs.
- Useful for safety-critical software like banking/financial, automobile, flight, and aerospace, defence applications.
- Many costly software failures could have been avoided with program verification: Ariane 5, Toyota sudden acceleration, power grid failure.

Motivating example

```
int max (int x, int y, int z) {
  if (x <= y) {
    if (y <= z) {
      return z;
    }
    else {
      return y;
    }
  }
  else {
    if (x <= z) {
      return y;
    }
    else {
      return x;
    }
  }
}
```

- Testing may not reveal a bug
- VCC verification tool reports an error

Hoare Logic

- A way of asserting properties of programs.
- Hoare triple: $\{A\}P\{B\}$ asserts that "Whenever program P is started in a state satisfying condition A, if it terminates, it will terminate in a state satisfying condition B."
- Example assertion: $\{m \geq 0\}\ P\ \{a = n * m\}$, where P is the program:

```
int a = 0;
int x = 0;
while (x < m) {
  a = a + n;
  x = x + 1;
}
```

- A proof system (due to Tony Hoare) for proving such assertions.
- A way of reasoning about such assertions using the notion of "Weakest Preconditions" (due to Dijkstra).

A simple programming language

- `skip`
- `x := e` (assignment)
- `if` b `then` S `else` T (if-then-else)
- `while` b `do` S (while)
- S `;` T (sequencing)

Example program

```
int a = 0;
int x = 0;
while (x < m) {
  a = a + n;
  x = x + 1;
}
```

Example program

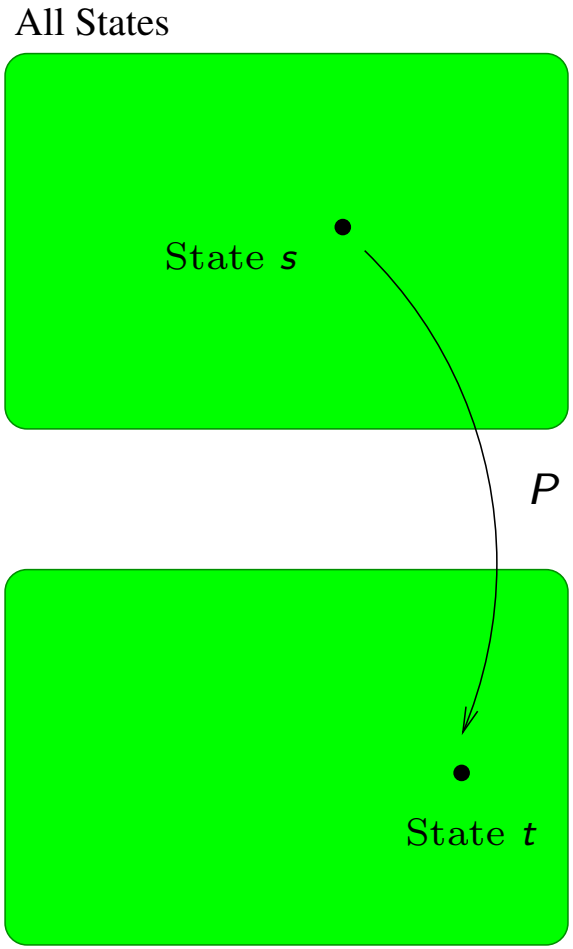
```
int a = 0;
int x = 0;
while (x < m) {
  a = a + n;
  x = x + 1;
}
```

```
S1: int a = 0;
S2: int x = 0;
S3: while (x < m) {
      a = a + n;
      x = x + 1;
    }
```

Program is `S1;S2;S3`

Programs as State Transformers

View program P as a partial map $[P] : \mathit{Stores} \to \mathit{Stores}$. (Assume that $\mathit{Stores} = \mathit{Var} \to \mathbb{Z}$.)

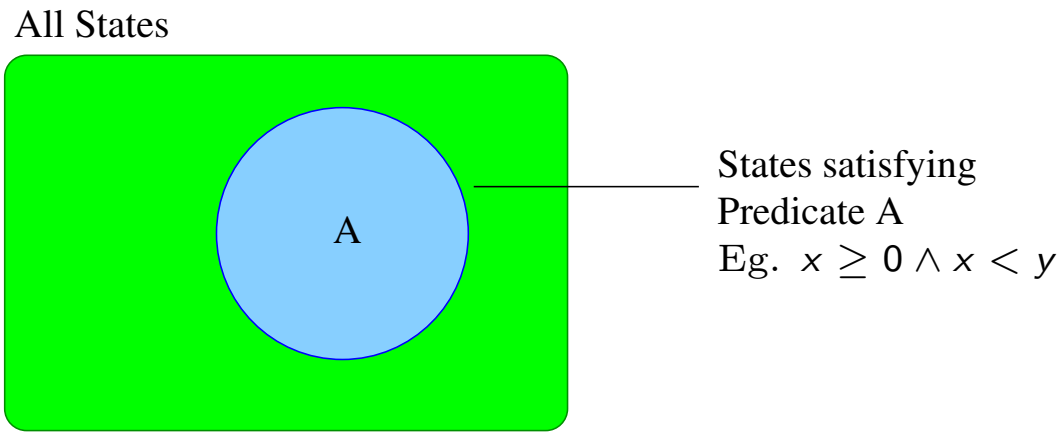


$\langle x \mapsto 2,\ y \mapsto 10,\ z \mapsto 3 \rangle$

```
y := y + 1;
z := x + y
```

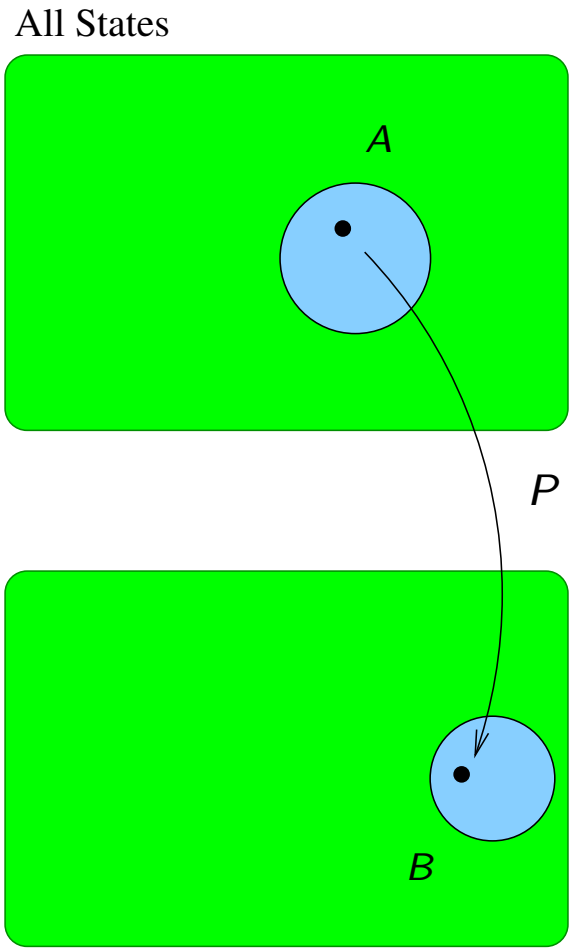
$\langle x \mapsto 2,\ y \mapsto 11,\ z \mapsto 13 \rangle$

Predicates on States



Assertion of "Partial Correctness" $\{A\}P\{B\}$

$\{A\}P\{B\}$ asserts that "Whenever program P is started in a state satisfying condition A, either it will not terminate, or it will terminate in a state satisfying condition B."



$\{10 \leq y\}$

```
y := y + 1;
z := x + y
```

$\{x < z\}$

Mathematical meaning of a Hoare triple

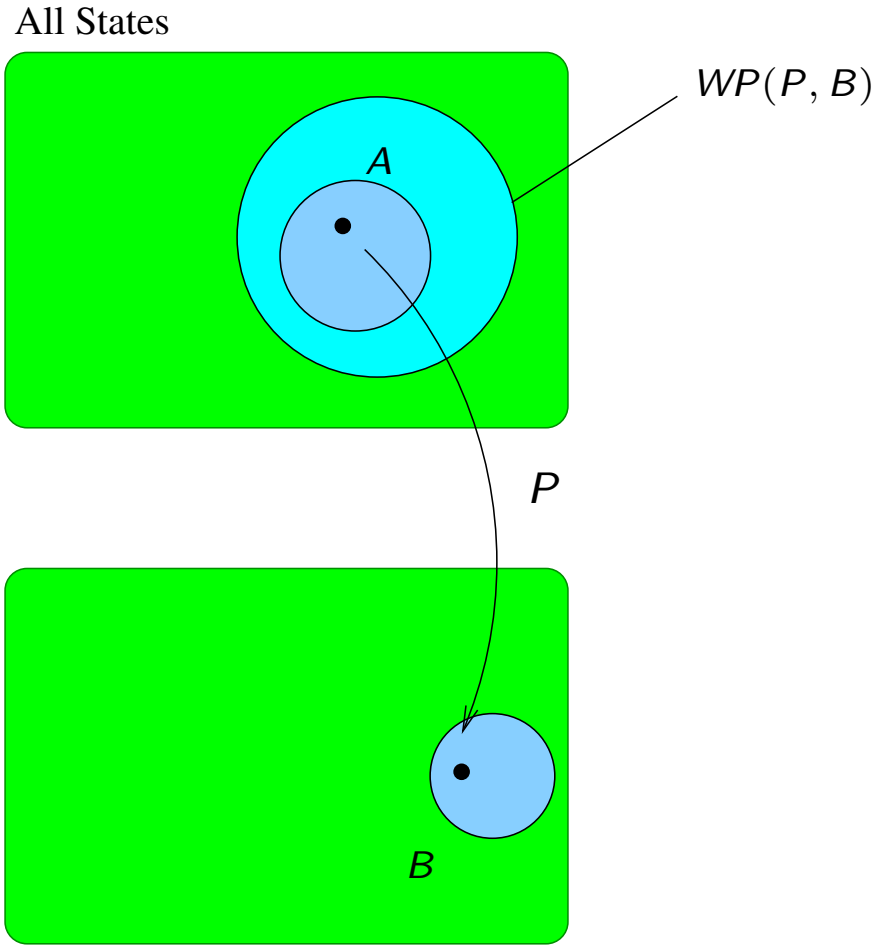
- View program P as a relation

$$[P] \subseteq \mathrm{Stores} \times \mathrm{Stores}.$$

  so that $(s, t) \in [P]$ iff it is possible to start P in the state s and terminate in state t.
- As usual here elements of Stores are maps from variables to integers.
- $[P]$ is possibly non-determinisitic, in case we also want to model non-deterministic assignment etc.
- Then the Hoare triple $\{A\}\ P\ \{B\}$ is true iff for all states s and t: whenever $s \models A$ and $(s, t) \in [P]$, then $t \models B$.
- In other words $Post_{[P]}([A]) \subseteq [B]$.

Weakest Precondition $WP(P, B)$

$WP(P, B)$ is "a predicate that describes the exact set of states s such that when program P is started in s, if it terminates it will terminate in a state satisfying condition B."



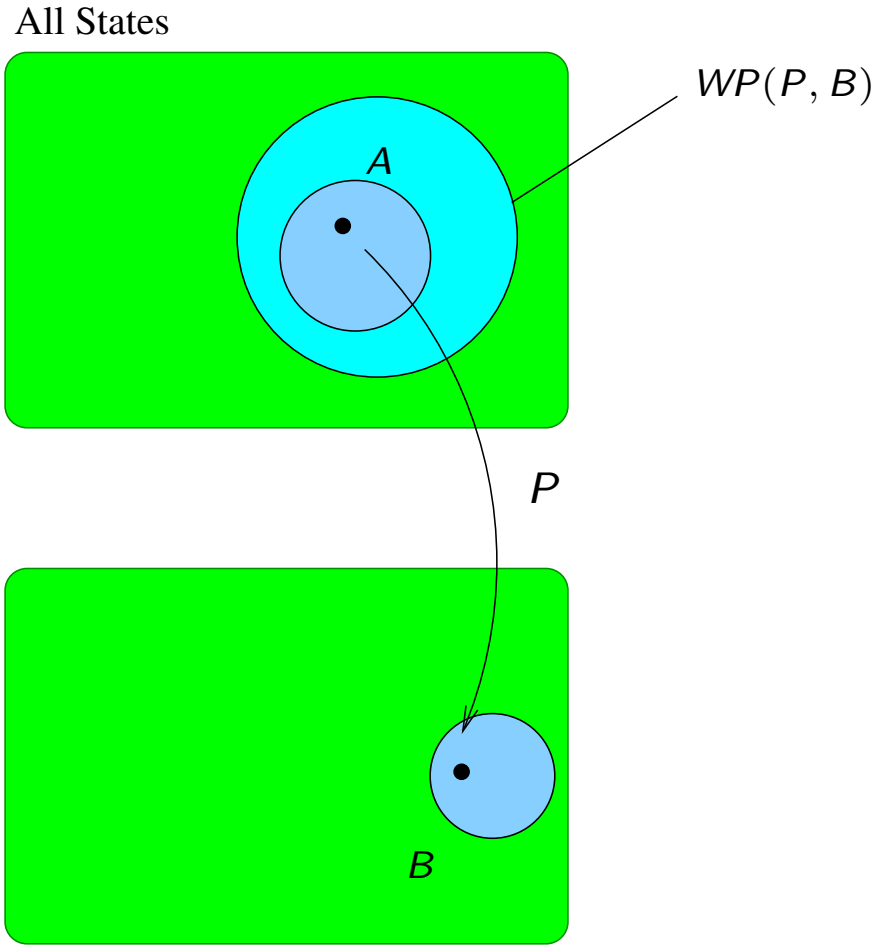
$\{10 < y\}$

```
y := y + 1;
z := x + y;
```

$\{x < z\}$

Weakest Precondition $WP(P, B)$

$WP(P, B)$ is "a predicate that describes the exact set of states s such that when program P is started in s, if it terminates it will terminate in a state satisfying condition B."



$\{-1 < y\}$

```
y := y + 1;
z := x + y;
```

$\{x < z\}$

Give "weakest" preconditions

1. $\{?\}$ `x := x + 2` $\{x \geq 5\}$

Give "weakest" preconditions

1. $\{\ x \geq 3\}$ `x := x + 2` $\{x \geq 5\}$

2. $\{?\qquad\qquad\qquad\qquad\qquad\qquad\}$
   `if (y < 0) then x := x+1 else x := y`
   $\{x > 0\}$

Give "weakest" preconditions

1. $\{\ x \geq 3\}$ `x := x + 2` $\{x \geq 5\}$

2. $\{\ (y < 0 \wedge x > -1) \vee (y > 0)\}$
   `if (y < 0) then x := x+1 else x := y`
   $\{x > 0\}$

3. $\{?\quad\quad\}$ while $(x \leq 5)$ do x := x+1 $\{x = 6\}$

Give "weakest" preconditions

1. $\{\ x \geq 3\}$ `x := x + 2` $\{x \geq 5\}$

2. $\{\ (y < 0 \wedge x > -1) \vee (y > 0)\}$
   `if (y < 0) then x := x+1 else x := y`
   $\{x > 0\}$

3. $\{\ x \leq 6\}$ while $(x \leq 5)$ do x := x+1 $\{x = 6\}$

Proof rules of Hoare Logic

Axiom of Valid formulas:

$$\frac{}{A}$$

provided "$\models A$" (i.e. A is a valid logical formula, eg. $x > 10 \implies x > 0$).

Skip:

$$\frac{}{\{A\}\ \texttt{skip}\ \{A\}}$$

Assignment

$$\frac{}{\{A[e/x]\}\ \texttt{x := e}\ \{A\}}$$

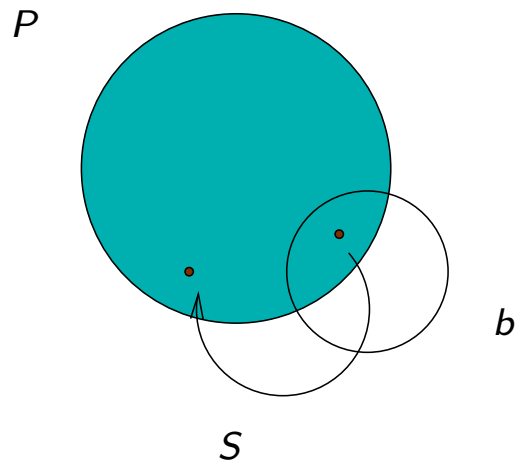
Proof rules of Hoare Logic

If-then-else:

$$\frac{\{P \wedge b\}\ S\ \{Q\},\ \{P \wedge \bar{b}\}\ T\ \{Q\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ T\ \{Q\}}$$

While (here P is called a *loop invariant*)

$$\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ \texttt{while}\ b\ \texttt{do}\ S\ \{P \wedge \bar{b}\}}$$



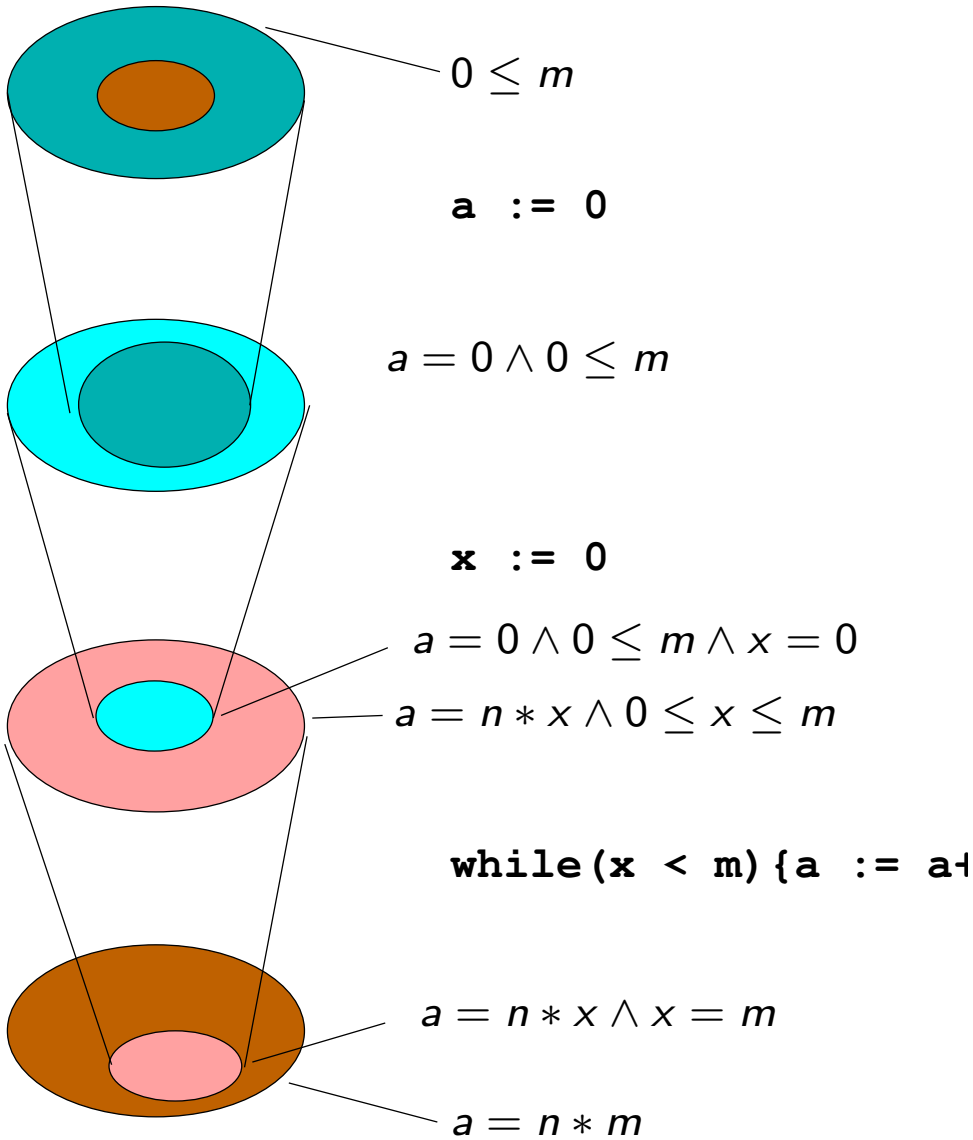
Sequencing:

$$\frac{\{P\}\ S\ \{Q\},\ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

Weakening:

$$\frac{P \Longrightarrow Q,\ \{Q\}\ S\ \{R\},\ R \Longrightarrow T}{\{P\}\ S\ \{T\}}$$

Idea of doing Hoare logic proofs



Some examples to work on

Use the rules of Hoare logic to prove the following assertions:

1. $\{x \geq 3\}$ `x := x + 2` $\{x \geq 5\}$
2. $\{(y \leq 0) \wedge (x > -1)\}$ `if` $(y < 0)$ `then x:=x+1 else x:=y` $\{x > 0\}$
3. $\{x \leq 0\}$ while $(x \leq 5)$ do x := x+1 $\{x = 6\}$

Soundness and Completeness of Hoare logic

- Hoare logic is sound (i.e. if we can prove "$\{A\}\ P\ \{B\}$" in the logic, then $\{A\}\ P\ \{B\}$ is true.)
- Conversely, is it "complete"? That is, if $\{A\}\ P\ \{B\}$ is true for a program P and pre/post-conditions A and B, does there exists a proof tree for $\{A\}\ P\ \{B\}$ using the rules of Hoare logic?
- Yes, provided the underlying logic L can express all "weakest preconditions" (for all programs and post-conditions expressed in L).

How VCC works: Generating Verification Conditions

To check:

$\{y > 10\}$

```
y := y + 1;
z := x + y;
```

$\{x < z\}$

Check verification condition:

$$(y > 10) \implies (y > -1).$$

by asking a theorem prover / SMT solver.