Floyd-Hoare Style Program Verification

Deepak D'Souza

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

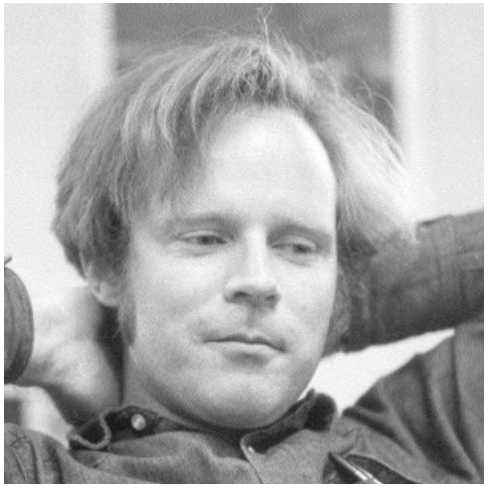
24 Feb 2020

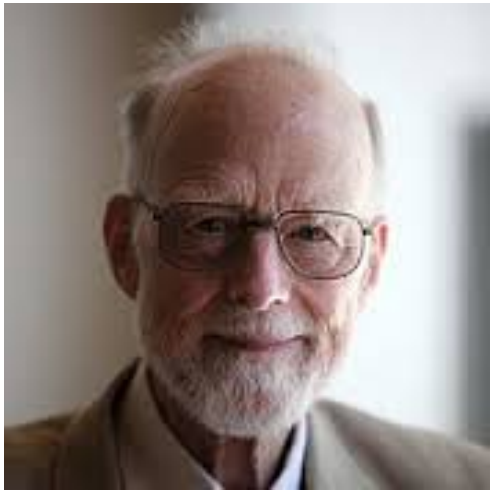
Outline of these lectures

Program Correctness

What does it mean for a program to be correct?

Floyd-Hoare Style of Program Verification





Robert W. Floyd: "Assigning meanings to programs" *Proceedings of the American Mathematical Society Symposia on Applied Mathematics* (1967)

C A R Hoare: "An axiomatic basis for computer programming", *Communications of the ACM* (1969).

Floyd-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: $\{n \geq 0\}\ P\ \{a = n + m\}$, where P is the program:

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

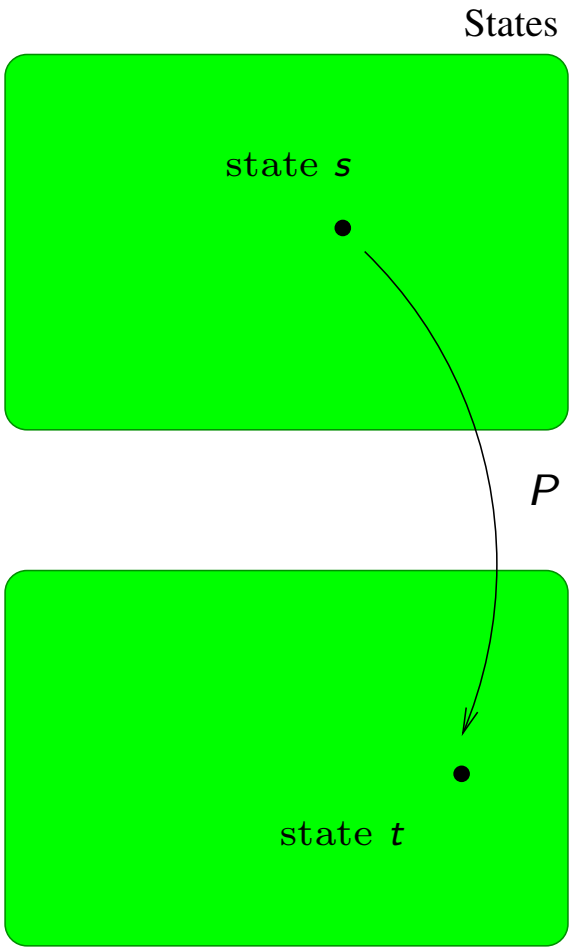
- Inductive Annotation ("consistent interpretation") (due to Floyd)
- A proof system (due to Hoare) for proving such assertions.
- A way of reasoning about such assertions using the notion of "Weakest Preconditions" (due to Dijkstra).

A simple programming language

- `skip` (do nothing)
- `x := e` (assignment)
- `if` b `then` S `else` T (if-then-else)
- `while` b `do` S (while loop)
- S ; T (sequencing)

Programs as State Transformers

- Program state is valuation to variables of the program: $States = Var \to \mathbb{Z}$.
- View program P as a partial map $[\![P]\!] : States \to States$.

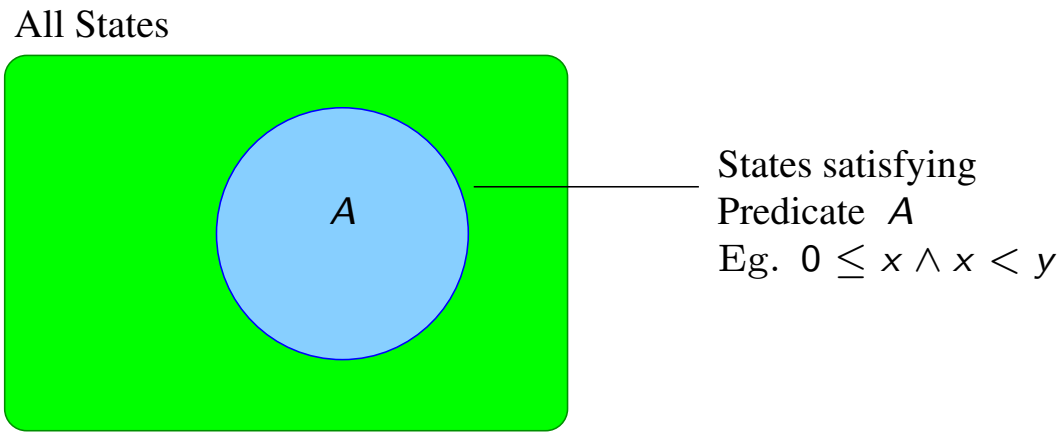


$s : \langle x \mapsto 2,\ y \mapsto 10,\ z \mapsto 3 \rangle$

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

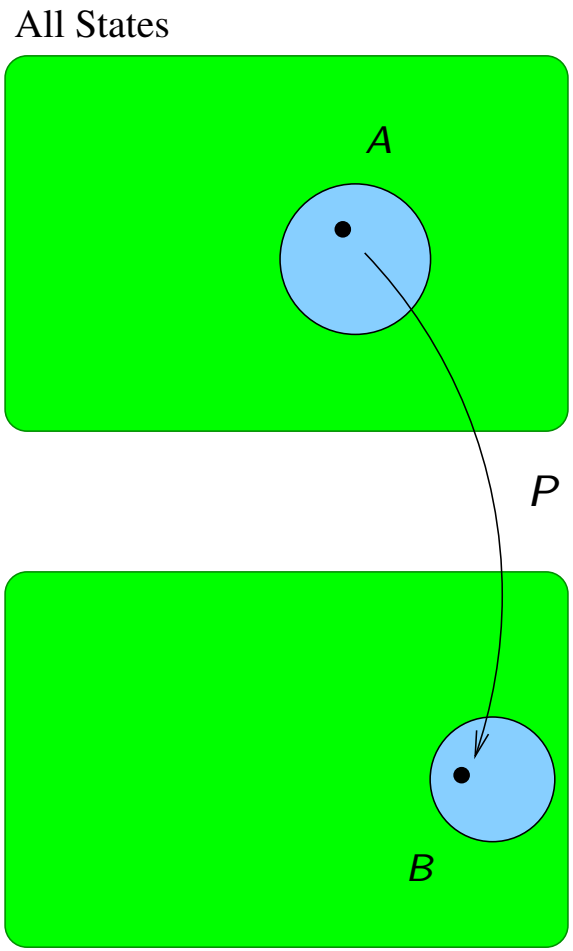
$t : \langle x \mapsto 2,\ y \mapsto 11,\ z \mapsto 13 \rangle$

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 on States (allows non-termination as well as non-determinism)

$$[\![P]\!] \subseteq \text{States} \times \text{States}.$$

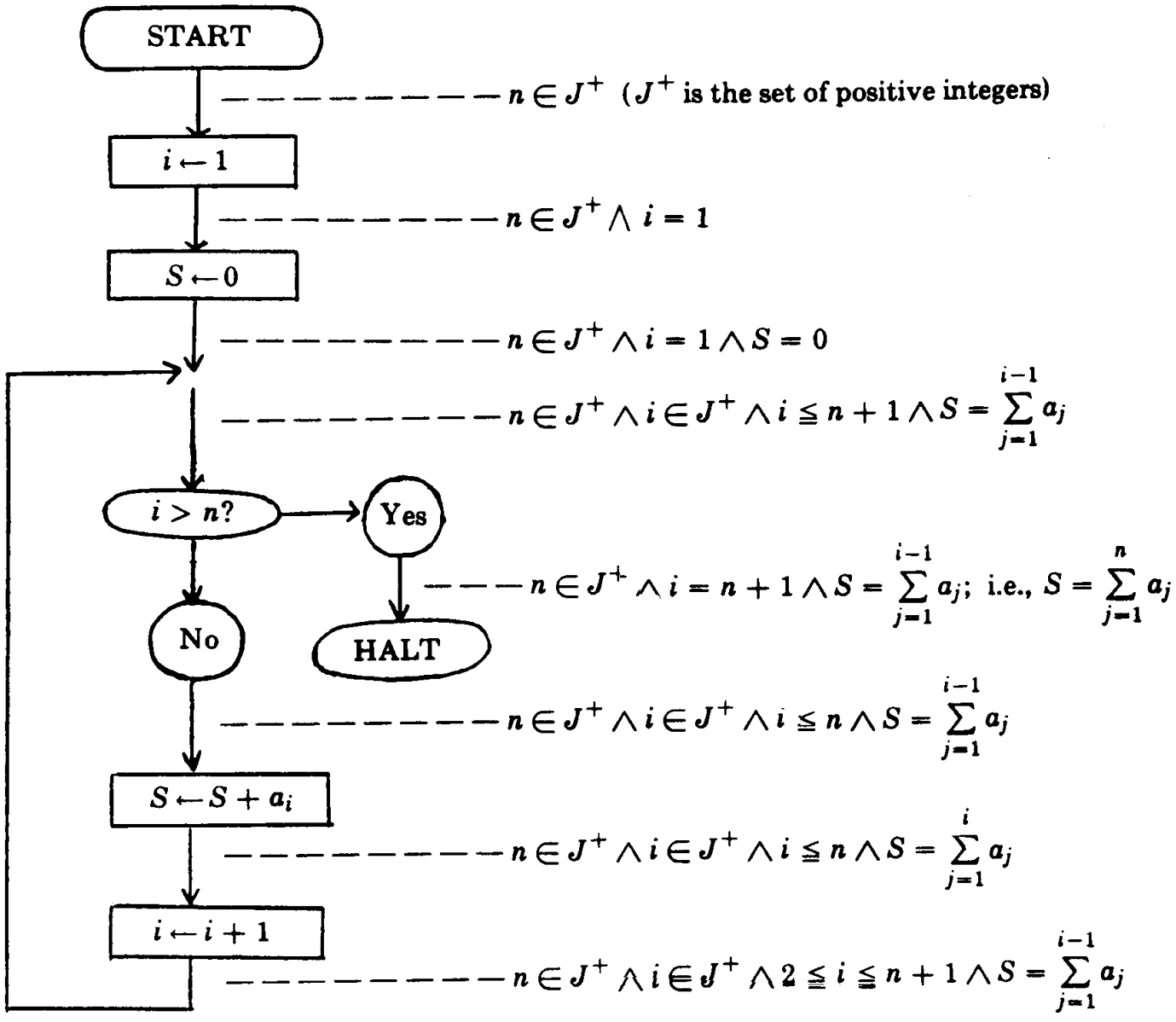
  Here $(s, t) \in [\![P]\!]$ iff it is possible to start P in the state s and terminate in state t.
- $[\![P]\!]$ is possibly non-determinisitic, in case we also want to model non-deterministic assignment etc.
- Then the Hoare triple $\{A\}\ P\ \{B\}$ is true iff for all states s and t: whenever $s \models A$ and $(s, t) \in [\![P]\!]$, then $t \models B$.
- In other words $Post_{[\![P]\!]}([\![A]\!]) \subseteq [\![B]\!]$.

Example programs and pre/post conditions

```
// Pre: true

if (a <= b)
  min := a;
else
  min := b;

// Post: min <= a && min <= b
```

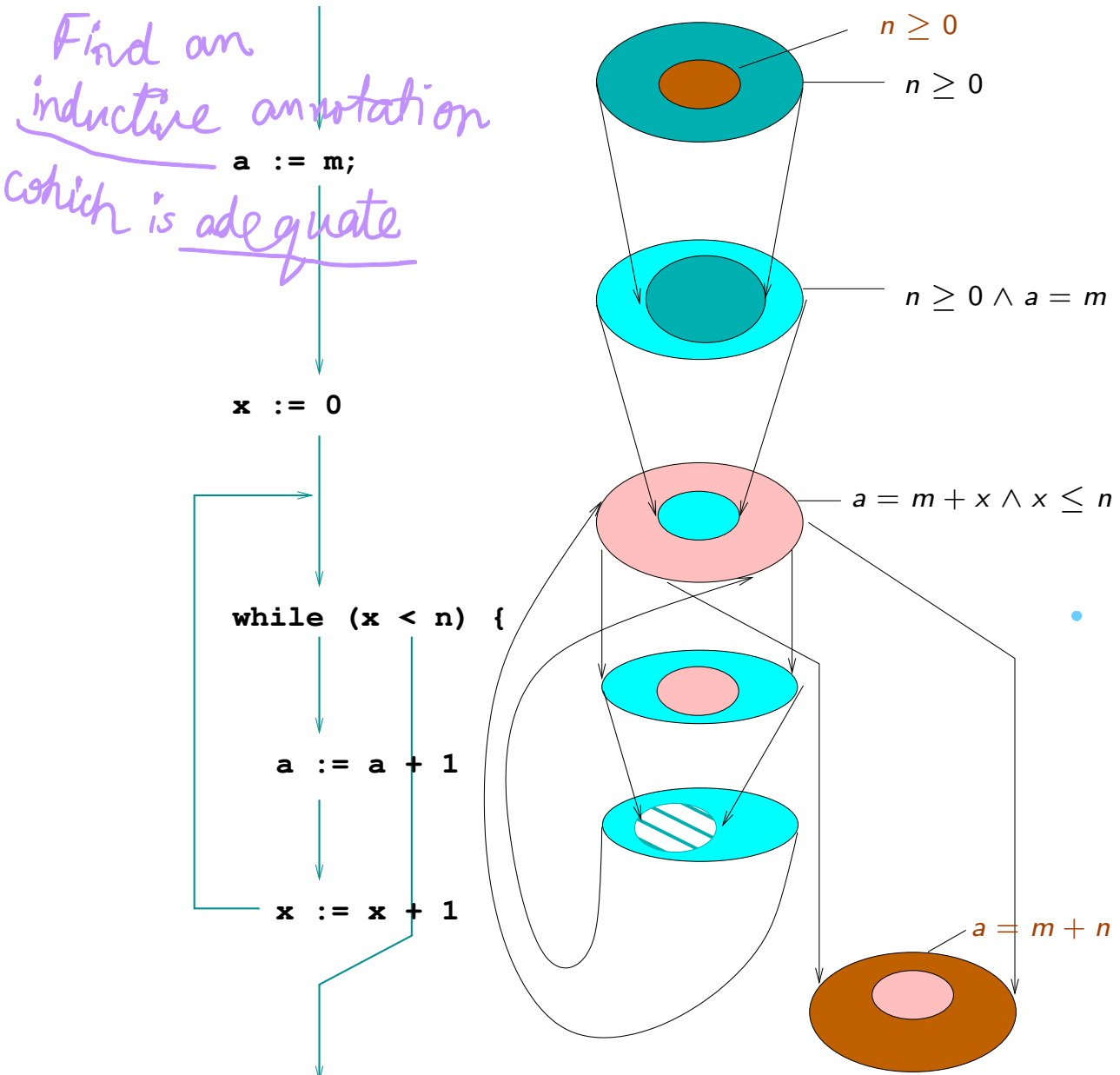
```
// Pre: 0 <= n

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

// Post: a = m + n
```

Floyd style proof: Inductive Annotation



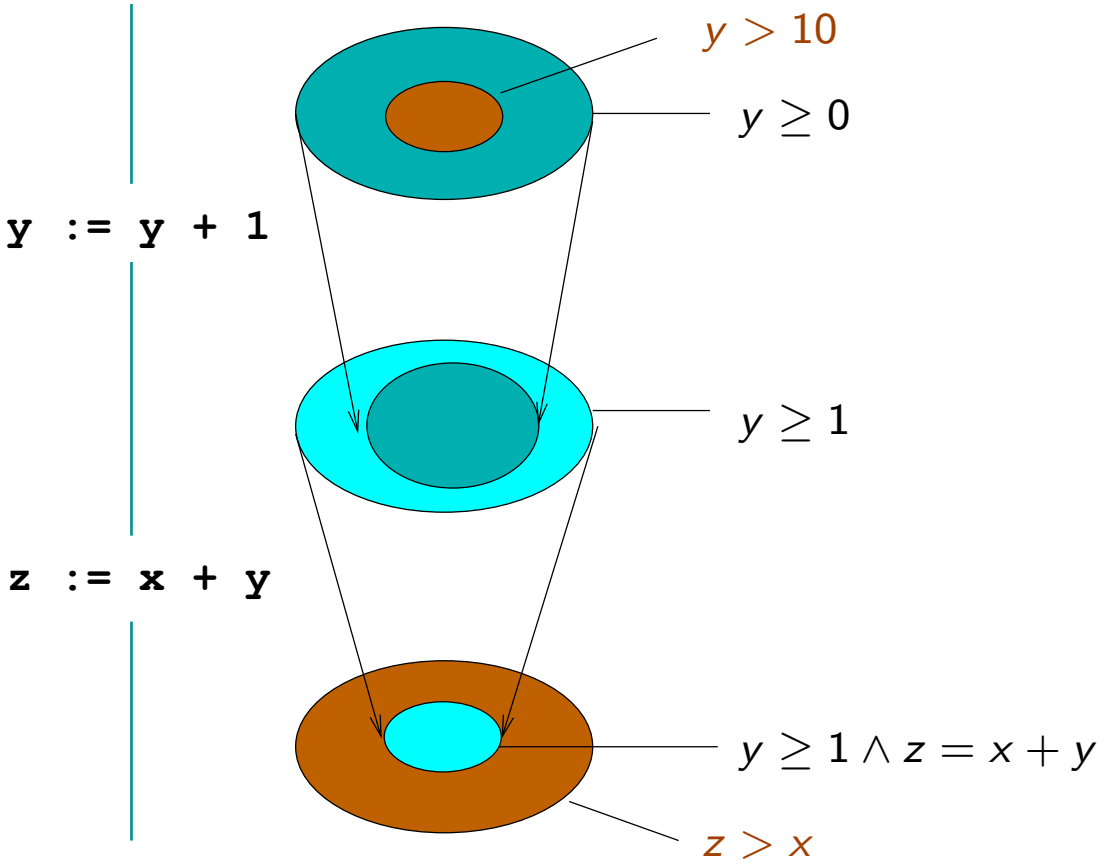
Inductive annotation based proof of a pre/post specification

- Annotate each program point i with a predicate A_i
- Successive annotations must be inductive:
  $[\![S_i]\!]([\![A_i]\!]) \subseteq [\![A_{i+1}]\!]$,
  OR logically:
  $A_i \wedge [S_i] \implies A'_{i+1}$.
- Annotation is adequate:
  $Pre \implies A_1$ and
  $A_n \implies Post$.
- Adequate annotation constitutes a proof of $\{Pre\}$ Prog $\{Post\}$.



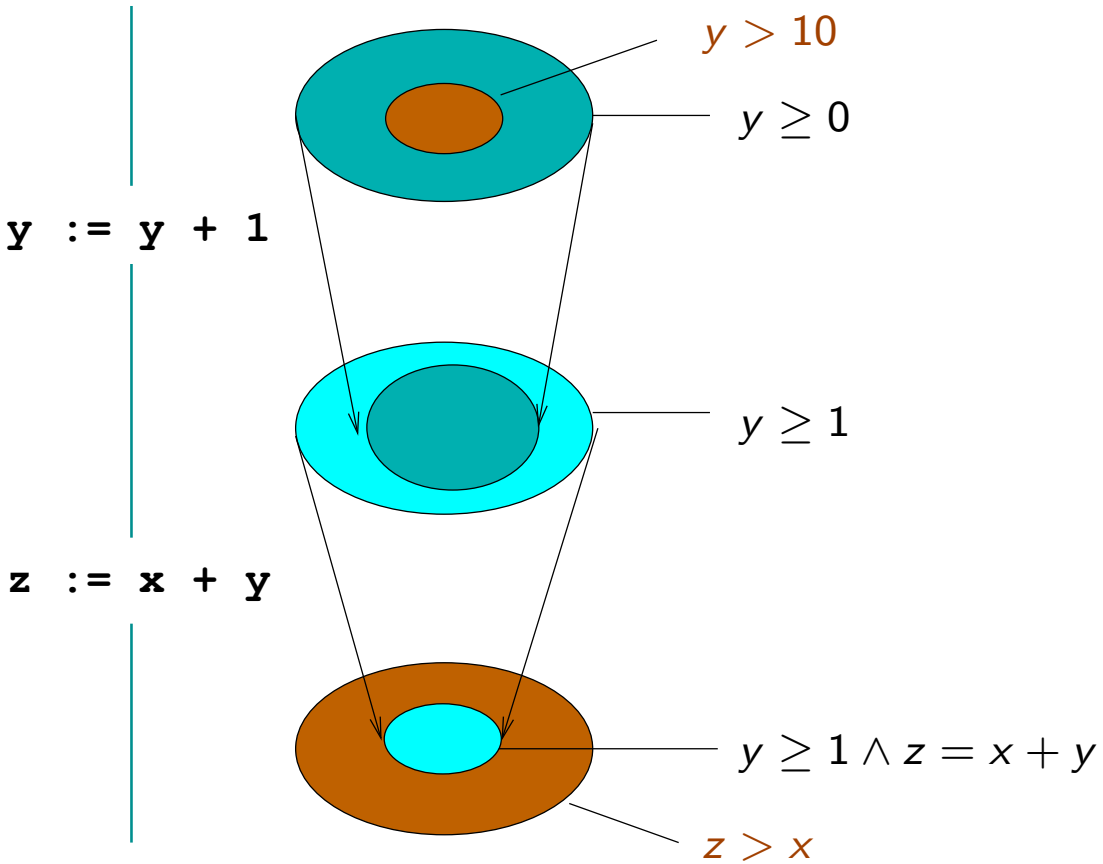
Example of inductive annotation

To prove: $\{y > 10\}$ `y := y+1; z := x+y` $\{z > x\}$



Example of inductive annotation

To prove: $\{y > 10\}$ `y := y+1; z := x+y` $\{z > x\}$



Logical proof obligations (VCs):

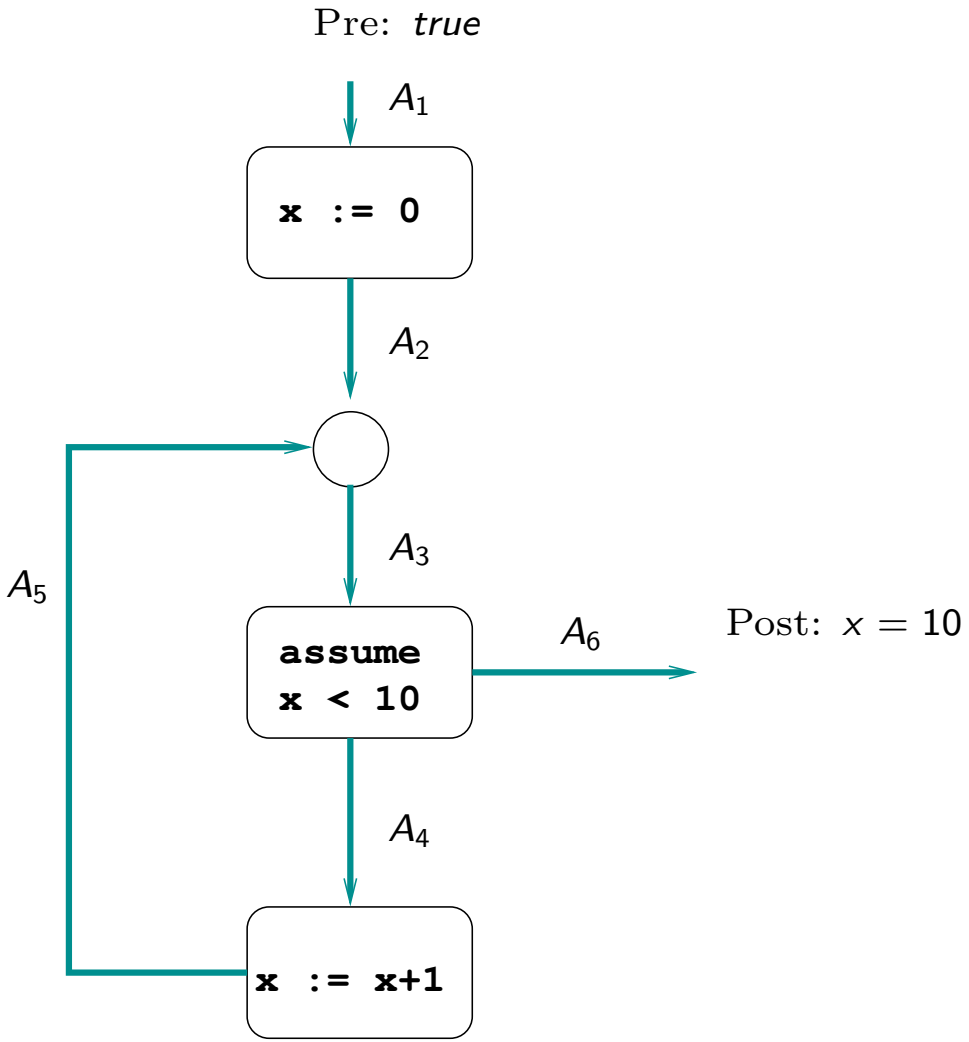
$(y > 10 \implies y \geq 0) \wedge ((y \geq 1 \wedge z = x + y) \implies z > x) \wedge$

$((y \geq 0 \wedge y' = y + 1) \implies y' \geq 1) \wedge ((y \geq 1 \wedge z' = x + y) \implies y' \geq$

Exercise 1

Prove using Floyd-style annotation:

```
// Pre: true
int x := 0;
while (x < 10)
  x := x + 1;
// Post: x = 10
```



Also write out the proof obligations (verification conditions).

Exercise 2

Prove using Floyd's inductive annotation:

$$\{n \geq 1\}\ P\ \{a = n!\},$$

where P is the program:

```
x := n;
a := 1;
while (x ≥ 1) {
    a := a * x;
    x := x - 1
}
```

Assume that factorial is defined as follows:

$$n! = \begin{cases} n \times (n-1) \times \cdots \times 1 & \text{if} \quad n \geq 1 \\ 1 & \text{if} \quad n = 0 \\ -1 & \text{if} \quad n < 0 \end{cases}$$

Exercise 2

Prove using Floyd's inductive annotation:

$$\{n \geq 1\}\ P\ \{a = n!\},$$

where P is the program:

```
S1: x := n;
S2: a := 1;
S3: while (x ≥ 1) {
S4:     a := a * x;
S5:     x := x - 1
    }
```

Assume that factorial is defined as follows:

$$n! = \begin{cases} n \times (n-1) \times \cdots \times 1 & \text{if} \quad n \geq 1 \\ 1 & \text{if} \quad n = 0 \\ -1 & \text{if} \quad n < 0 \end{cases}$$

Hoare's view: Program as a composition of statements

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

Hoare's view: Program as a composition of statements

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

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

Program is S1;S2;S3

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:

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

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

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

Sequencing:

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

Weakening:

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

Loop invariants

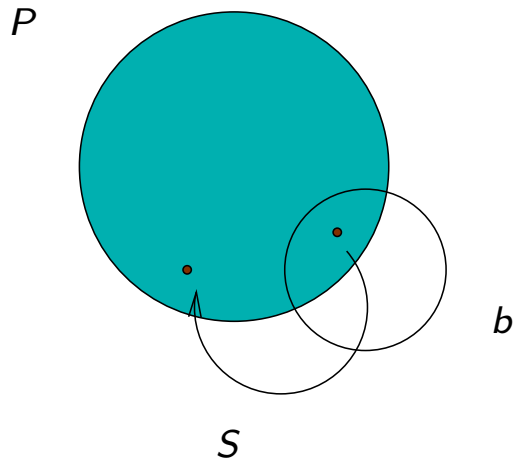
A predicate P is a loop invariant for the while loop:

```
while (b) {
  S
}
```

if $\{P \wedge b\}\ S\ \{P\}$ holds.

If P is a loop invariant then we can infer that:

$$\{P\}\ \texttt{while}\ b\ \texttt{do}\ S\ \{P \wedge \neg b\}$$



Some examples to work on

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

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

Example proof using Hoare Logic

1. $\{n \geq 0\}$ S1 $\{n > 0 \wedge a = m\}$
2. $\{n \geq 0 \wedge a = m\}$ S2 $\{n \geq 0 \wedge a = m \wedge x = 0\}$
3. $\{a = m + x \wedge 0 \leq x \leq n \wedge x < n\}$ S4;S5 $\{a = m + x \wedge 0 \leq x \leq n\}$
4. $\{a = m + x \wedge 0 \leq x \leq n\}$ S3 $\{a = m + x \wedge 0 \leq x \leq n \wedge x \geq n\}$
5. $\{n \geq 0\}$ S1;S2 $\{n \geq 0 \wedge a = m \wedge x = 0\}$ (From Seq rule, 1 and 2)
6. $(n \geq 0 \wedge a = m \wedge x = 0) \implies (a = m + x \wedge 0 \leq x \leq n)$ (From logical axiom)
7. $\{n \geq 0\}$ S1;S2 $\{a = m + x \wedge 0 \leq x \leq n\}$ (From Weakening rule, 5 and 6)
8. $\{n \geq 0\}$ (S1;S2);S3 $\{a = m + x \wedge 0 \leq x \leq n \wedge x \geq n\}$ (From Seq rule, 7, 4)
9. $(a = m + x \wedge 0 \leq x \leq n \wedge x \geq n) \implies (a = m + n)$
10. $\{n \geq 0\}$ (S1;S2);S3 $\{a = m + n\}$ (From Weakening rule, 8, 9).

$A = n \geq 0 \wedge a = m$

$A(e/x) = n \geq 0 \wedge m = m$

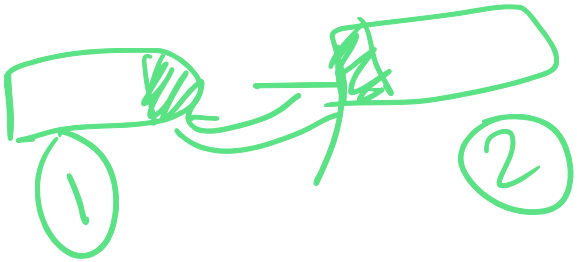
```
// pre: n >= 0
S1: int a := m;
S2: int x := 0;
S3: while (x < n) {
S4:    a := a + 1;
S5:    x := x + 1;
    }
// post: a = m + n
```

Program is S1;S2;S3

P

1, 2, 5

3, 4



```
S1  q := 0;
S2  r := x;
S3  while (r >= y) {
S4    r := r - y;
S5    q := q + 1;
    }
```

S_3 while b do S_1

$\{x \geq 0 \wedge y \geq 0\}\ P\ \{q \cdot y + r = x \wedge 0 \leq r < y\}.$

$\wedge$ | $\wedge$

① $\{x \geq 0 / y \geq 0\}\ S_1\ \{x \geq 0, y \geq 0 / q = 0\}$

② $\{x \geq 0 / y \geq 0, q = 0\}\ S_2\ \{x \geq 0 / y \geq 0, q = 0, r = x\}$

① ⊕ ②

③ $\{x \geq 0 / y \geq 0\}\ S_1 S_2\ \{x \geq 0 / y \geq 0, q = 0 / r = x\}$

④ $\{x = q \cdot y + r\ /\ r \geq y\}\ S_4 S_5$

$\{x = q \cdot y + r\}$

$\{P \wedge b\}\ S\ \{P\}$ — P while b

⑤ $\{x = q y + r\}\ S_3\ \{x = qy + r / r < y\}$

⑥ $\{x \geq 0 / y \geq 0 / q = 0 / r = x\} \Rightarrow \{x = q y + r\}$

⑦ $\{x \geq 0 / y \geq 0\}\ S_1 S_2 S_3\ \{x = qy + r / r < y\}$

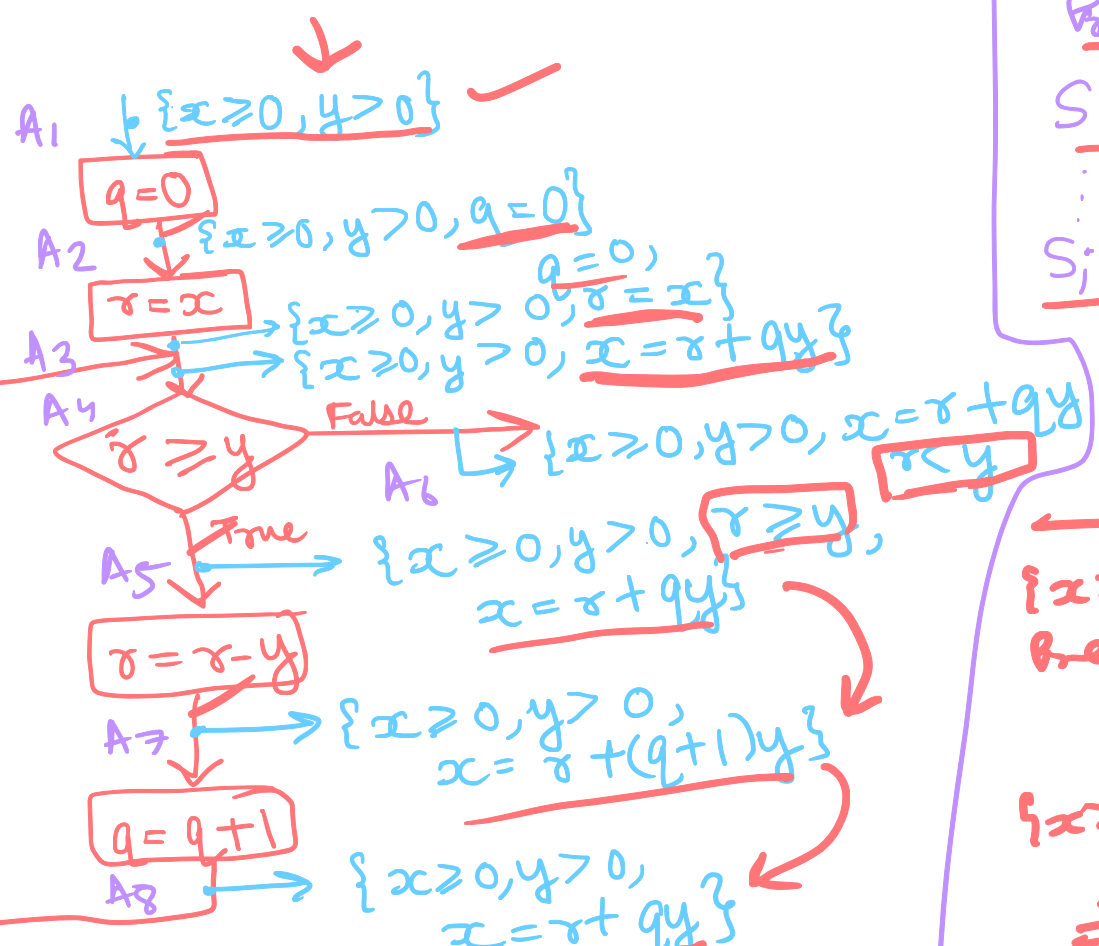
⑧ $\{x \geq 0 / y \geq 0\}\ sP\ \{x = qy + r \wedge r < y\}$

```
q := 0;
r := x;
while (r >= y) {
  r := r - y;
  q := q + 1;
}
```

Floyd

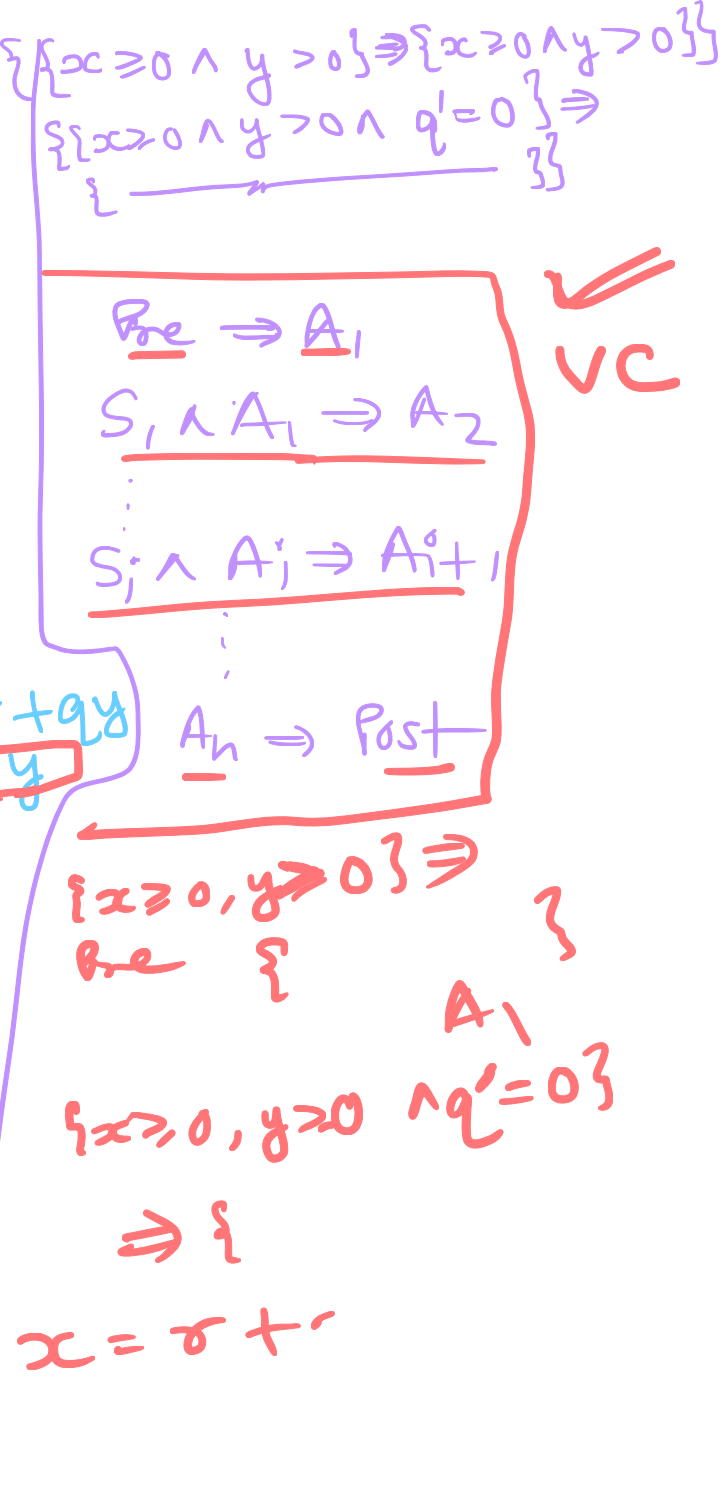
$\{x \geq 0 \wedge y \geq 0\}\ P\ \{q \cdot y + r = x \wedge 0 \leq r < y\}.$

Pre — Post



A1

A4 A5
A5 A1



a —
b — VC

$x = x + 1$

$x' = 0$

$a \wedge b \longrightarrow x' = x + 1$

Exercise

Prove using Hoare logic:

$$\{n \geq 1\}\ P\ \{a = n!\},$$

where P is the program:

```
x := n;
a := 1;
while (x ≥ 1) {
    a := a * x;
    x := x - 1
}
```

Assume that factorial is defined as follows:

$$n! = \begin{cases} n \times (n-1) \times \cdots \times 1 & \text{if} \quad n \geq 1 \\ 1 & \text{if} \quad n = 0 \\ -1 & \text{if} \quad n < 0 \end{cases}$$

Exercise

Prove using Hoare logic:

$$\{n \geq 1\}\ P\ \{a = n!\},$$

where P is the program:

```
S1: x := n;
S2: a := 1;
S3: while (x ≥ 1) {
S4:     a := a * x;
S5:     x := x - 1
    }
```

Assume that factorial is defined as follows:

$$n! = \begin{cases} n \times (n-1) \times \cdots \times 1 & \text{if} \quad n \geq 1 \\ 1 & \text{if} \quad n = 0 \\ -1 & \text{if} \quad n < 0 \end{cases}$$

```
S1: x := n;
S2: a := 1;
S3: while (x ≥ 1) {
S4:     a := a * x;
S5:     x := x - 1
    }
```

$\{n \geq 1\}\ P\ \{a = n!\},$

a	x
1	4
4	3
3	2
2	1
1	0

$\{n \geq 1\}\ S_1 S_2\ \{n \geq 1 \ / \ x = n \ / \ a = 1\}$

$\{a = \prod_{i=n}^{x+1} i \ / \ x \geq 1\}\ S_4 S_5\ \{a = \prod_{i=n}^{x+1} i\}$

$x+1 \leq n$

$\{a = \prod_{i=n}^{x+1} i\}\ S_3\ \{a = \prod_{i=n}^{x+1} i \ / \ x < 1\}$

$x+1 \leq n$

$x+1 \leq n$

$a =$

Soundness and Completeness

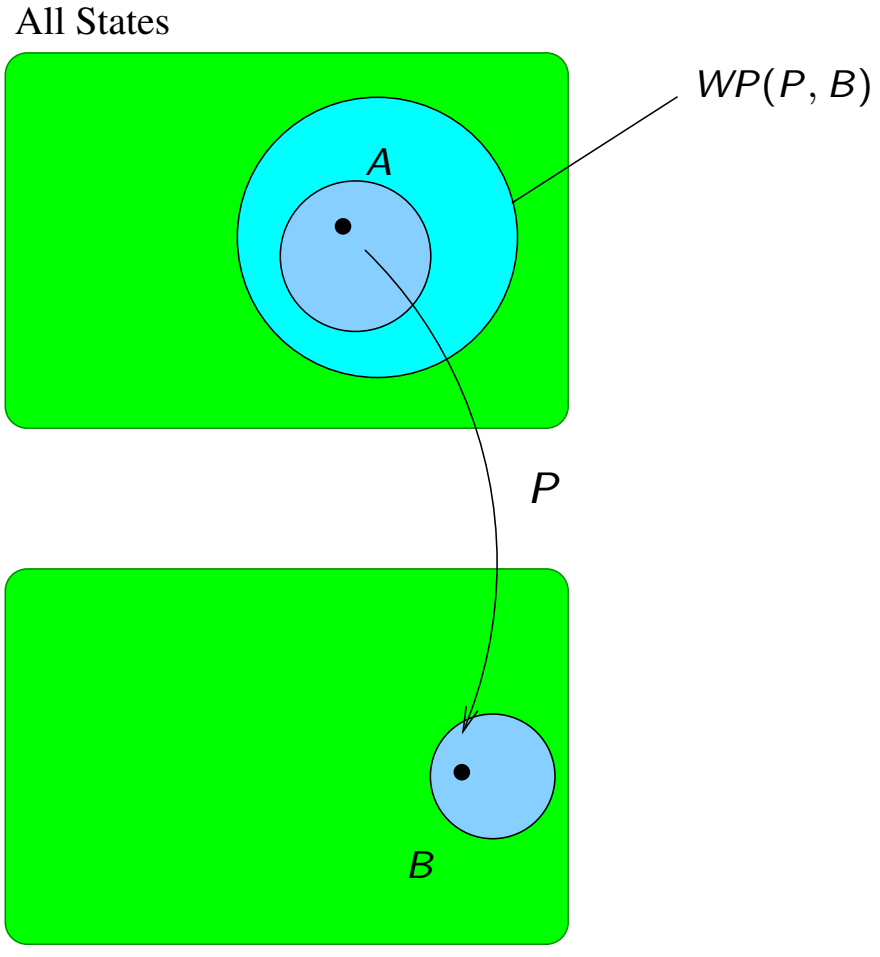
Soundness: If our proof system proves $\{A\}\ P\ \{B\}$ then $\{A\}\ P\ \{B\}$ indeed holds.

Completeness: If $\{A\}\ P\ \{B\}$ is true then our proof system can prove $\{A\}\ P\ \{B\}$.

- Floyd proof style is sound since any execution must stay within the annotations. Complete because the "collecting" set is an adequate inductive annotation for any program and any true pre/post condition.
- Hoare logic is sound, essentially because the individual rules can be seen to be sound.
- For completness of Hoare logic, we need weakest preconditions.

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;
```

$\{x < z\}$

Exercise: Give "weakest" preconditions

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

Exercise: 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`
   $\{x > 0\}$

Exercise: 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` $\{x = 6\}$

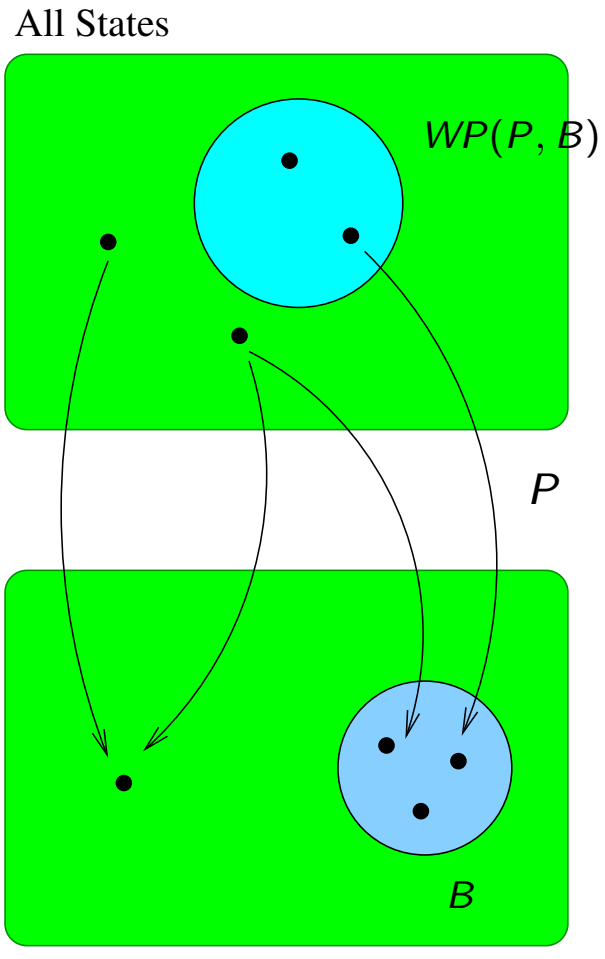
Exercise: Give "weakest" preconditions

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

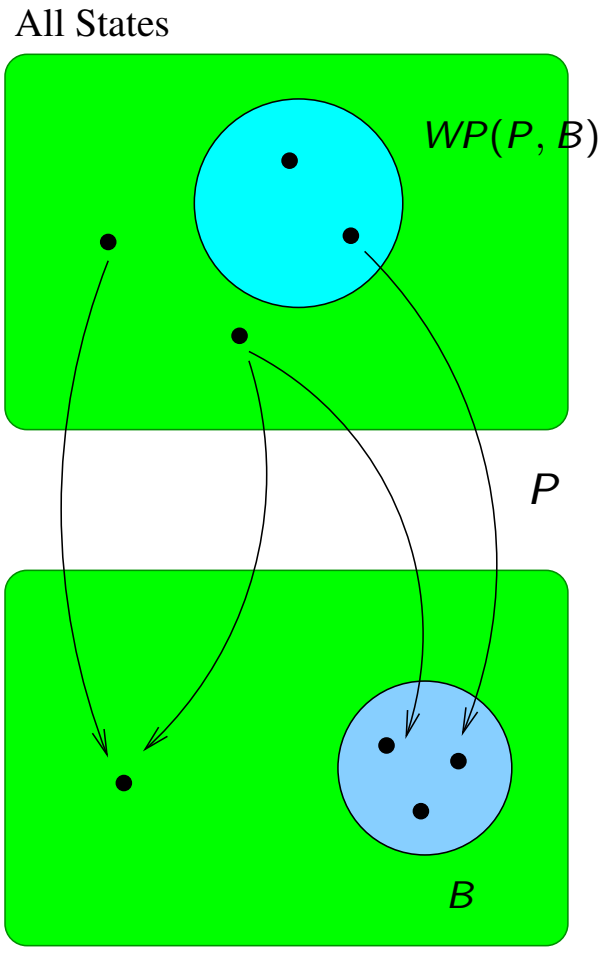
2. $\{\ (y < 0 \wedge x > -1) \vee (y > 0)\}$
   `if (y < 0) then x := x+1 else x := y`
   $\{x > 0\}$

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

Exercise: How will you define $WP(P, B)$?



Exercise: How will you define $WP(P, B)$?



$$WP(P, B) = \{s \mid \forall t : (s, t) \in [P] \text{ we have } t \models B\}$$

Using weakest preconditions to partially automate inductive proofs

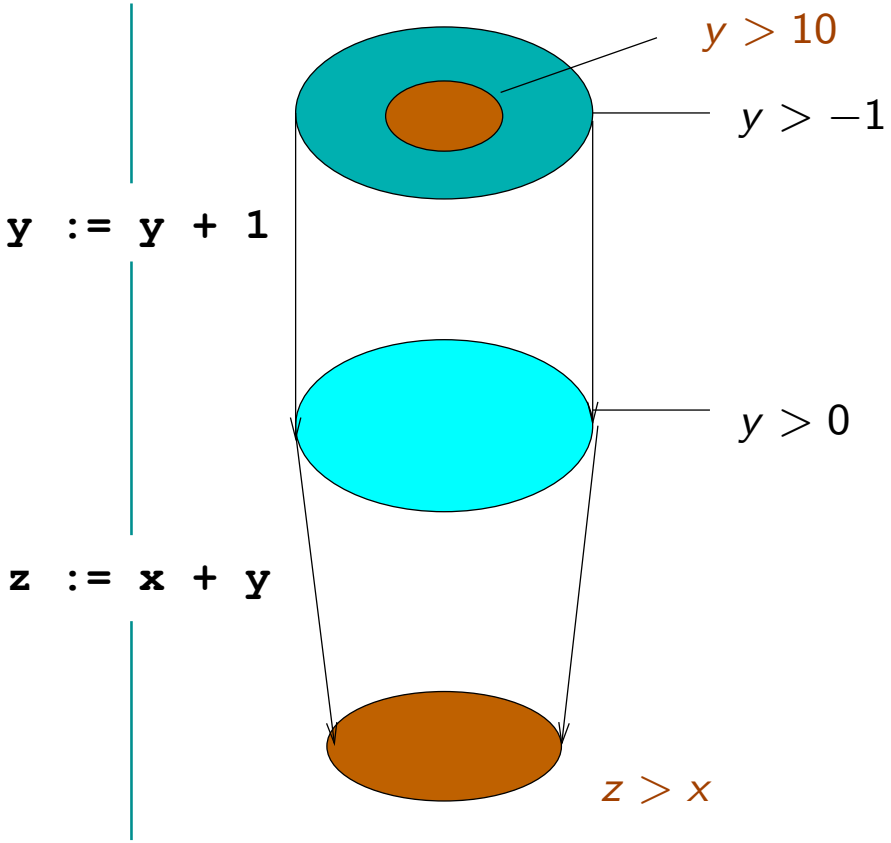
Weakest preconditions give us a way to:

- Check inductiveness of annotations

$$\{A_i\}\ S_i\ \{A_{i+1}\} \text{ iff } A_i \implies WP(S_i, A_{i+1})$$

- Reduce the amount of user-annotation needed
  - Programs without loops don't need any user-annotation
  - For programs with loops, user only needs to provide loop invariants

Checking $\{A\}\ P\ \{B\}$ using WP



Check that

$$(y > 10) \implies WP(P, z > x)$$

WP rules

- Hoare's rules for skip, assignment, and if-then-else are already WP rules.
- For Sequencing:

$$WP(S;T,\ B) = WP(S, WP(T, B)).$$

Weakest Precondition for `while` statements

- We can "approximate" $WP(\texttt{while}\ b\ \texttt{do}\ c)$.
- $WP_i(w, A)$ = the set of states from which the body c of the loop is either entered more than i times or we exit the loop in a state satisfying A.
- WP_i defined inductively as follows:

$$\begin{aligned} WP_0 &= b \vee A \\ WP_{i+1} &= (\neg b \wedge A) \vee (b \wedge WP(c, WP_i)) \end{aligned}$$

- Then $WP(w, A)$ can be shown to be the "limit" or least upper bound of the chain $WP_0(w, A)$, $WP_1(w, A), \ldots$ in a suitably defined lattice (here the join operation is "And" or intersection).

Illustration of WP_i through example

Consider the program w below:

```
while (x ≥ 10) do
    x := x - 1
```

- What is the weakest precondition of w with respect to the postcondition $(x \leq 0)$?
- Compute $WP_0(w, (x \leq 0))$, $WP_1(w, (x \leq 0))$, ...

Illustration of WP_i through example

Consider the program w below:

```
while (x ≥ 10) do
    x := x - 1
```

- What is the weakest precondition of w with respect to the postcondition $(x \leq 0)$?
- Compute $WP_0(w, (x \leq 0))$, $WP_1(w, (x \leq 0))$, ...

Automating checking of pre-post specifications for a program

To check:

$\{y > 10\}$

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

$\{x < z\}$

Use the weakest precondition rules to generate the verification condition:

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

Check the verification condition by asking a theorem prover / SMT solver if the formula

$$(y > 10) \wedge \neg(y > -1).$$

is satisfiable.

What about `while` loops?

```
Pre: 0 <= n

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

Post: a = m + n
```

Adequate loop invariant

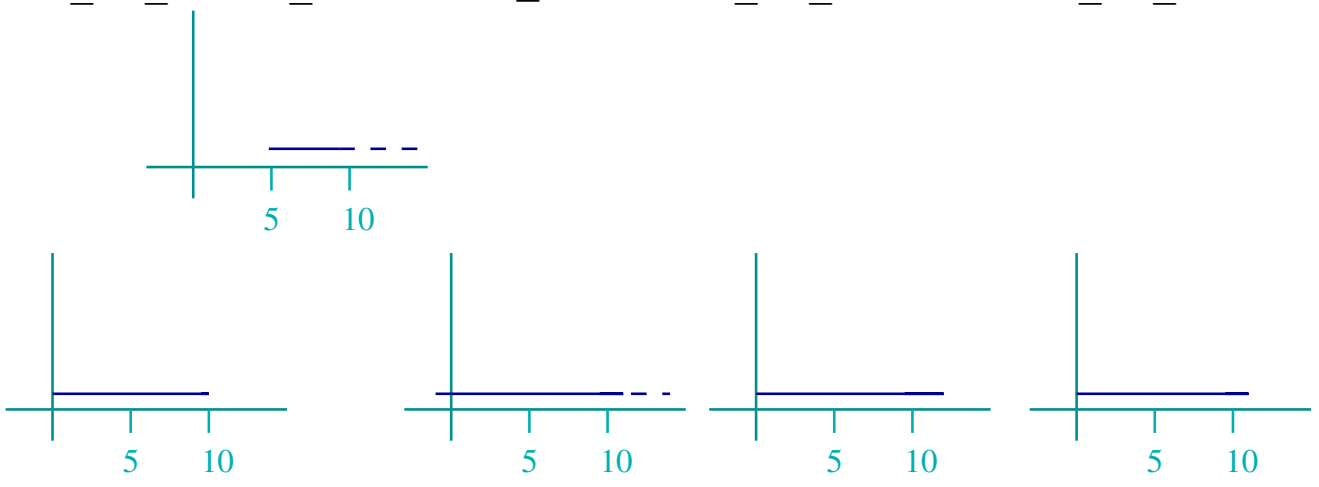
What is a "good" loop invariant for this program?

```
x := 0;
while (x < 10) {
 if (x >= 0)
  x := x + 1;
 else
  x := x - 1;
}
assert(x <= 11);
```

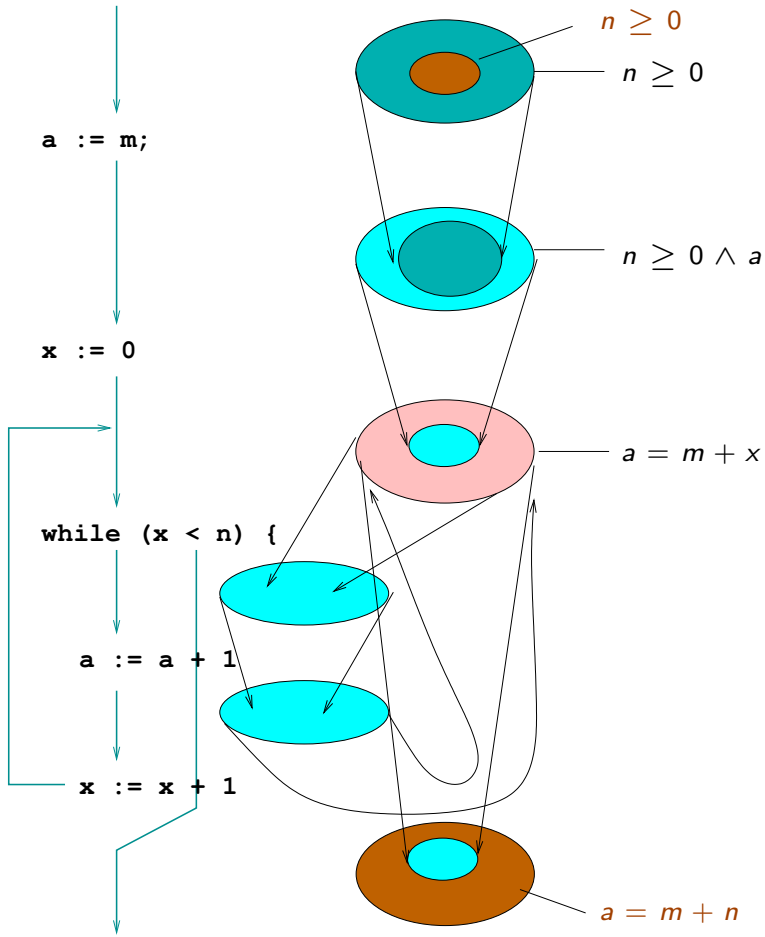
Adequate loop invariant

```
x := 0;
while (x < 10) {
 if (x >= 0)
  x := x + 1;
 else
  x := x - 1;
}
assert(x <= 11);
```

Canonical Invariant	*Not−inv*	*Inv,not−ind*	*Inv,ind,not−adeq*	*Inv,ind,adeq*
$0 \leq x \leq 10$	$5 \leq x$	$-1 \leq x$	$0 \leq x \leq 12$	$0 \leq x \leq 11$



Adequate loop invariant



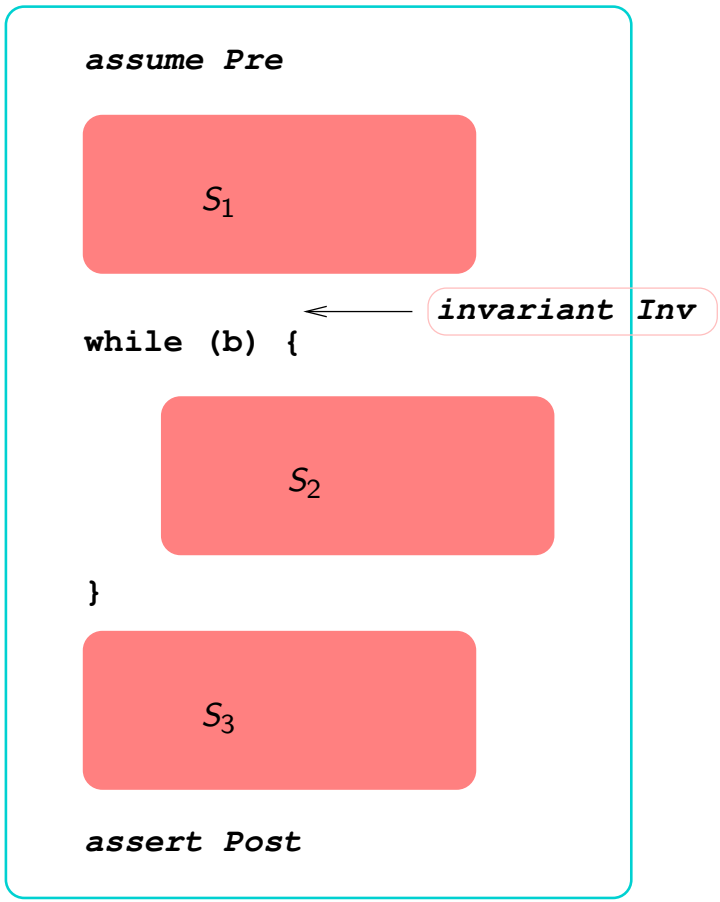
An adequate loop invariant needs to satisfy:

- $\{n \geq 0\}$ `a := m; x := 0` $\{a = m + x \wedge x \leq n\}$.
- $\{a = m + x \wedge x \leq n \wedge x < n\}$ `a := a+1; x := x+1` $\{a = m + x \wedge x \leq n\}$.
- $\{a = m + x \wedge x \leq n \wedge x \geq n\}$ `skip` $\{a = m + n\}$.

Verification conditions are generated accordingly.

Note that $a = m + x$ is not an adequate loop invariant.

Generating Verification Conditions for a program



The following VCs are generated:

- $Pre \wedge [S_1] \implies Inv'$
  Or: $Pre \implies WP(S_1, Inv)$
- $Inv \wedge b \wedge [S_2] \implies Inv'$
  Or: $(Inv \wedge b) \implies WP(S_2, Inv)$
- $Inv \wedge \neg b \wedge [S_3] \implies Post'$
  Or: $Inv \wedge \neg b \implies WP(S_3, Post)$

Relative completeness of Hoare logic

Theorem (Cook 1974)

Hoare logic is complete provided the assertion language L can express the WP for any program P and post-condition B.

Proof uses WP predicates and proceeds by induction on the structure of the program P.

- Suppose $\{A\}$ `skip` $\{B\}$ holds. Then it must be the case that $A \implies B$ is true. By Skip rule we know that $\{B\}$ `skip` $\{B\}$. Hence by Weakening rule, we get that $\{A\}$ `skip` $\{B\}$ holds.
- Suppose $\{A\}$ `x := e` $\{B\}$ holds. Then it must be the case that $A \implies B[e/x]$. By Assignment rule we know that $\{B[e/x]\}$ `x := e` $\{B\}$ is true. Hence by Weakening rule, we get that $\{A\}$ `x := e` $\{B\}$ holds.
- Similarly for sequencing `S;T`.
- Similarly for `if-then-else`.

Relative completeness of Hoare logic

- Suppose $\{A\}$ `while b do S` $\{B\}$ holds. Let $P = WP(\texttt{while b do S}, B)$. Then it is not difficult to check that P is a loop invariant for the while statement. I.e $\{P \wedge b\}$ `S` $\{P\}$ is true. By induction hypothesis, this triple must be provable in Hoare logic. Hence we can conclude using the While rule, that $\{P\}$ `while b do S` $\{P \wedge \neg b\}$. But since P was a valid precondition, it follows that $(P \wedge \neg b) \implies B$. For the same reason, we should have $A \implies P$. By the weakening rule, we have a proof of $\{A\}$ `while b do S` $\{B\}$.

Conclusion

- Features of this Floyd-Hoare style of verification:
  - Tries to find a proof in the form of an inductive annotation.
  - A Floyd-style proof can be used to obtain a Hoare-style proof; and vice-versa.
  - Reduces verification (given key annotations) to checking satisfiability of a logical formula (VCs).
  - Is flexible about predicates, logic used (for example can add quantifiers to reason about arrays).
- Main challenge is the need for user annotation (adequate loop invariants).
- Can be increasingly automated (using learning techniques).

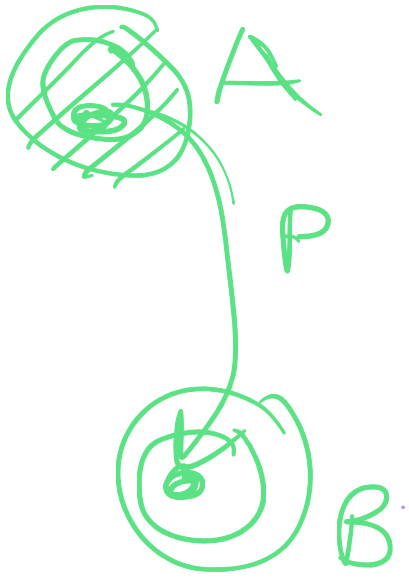
1. Give the weakest precondition for

$x \leq 10$

```
do
   x := x+1;
while (x <= 10)
```

$x > 10$

2. Describe $WP(do\ S while(b))$ in terms of $WP(while(b)\ do\ S)$.



S ; while(b) do S

S_1 S_2

$(S, WP(while(b)\ do\ S))$

$W(P, B) = A$