19CSE205 Program Reasoning Iterative structures

Dr.S.Padmavathi

CSE, Amrita School of Engineering

Coimbatore

Program Verification

Objective: To prove that a program P is correct with respect to its contract which is stated as a pre-condition I and post-condition O.

The Weakest Precondition of a statement S w.r.t. a post-condition O is written as wp(S, O).

If the input condition for program P is I, then we want the following theorem to be true:

$$I \Longrightarrow wp(P, O)$$

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Defining Weakest Preconditions

1. wp(x = expr, O).

2. wp(S1 ; S2, O).

3a. wp(if (B) S1 else S2, O).
3b. wp(if (B) S1, O).

4. wp(while B do S, O).

Assignment

Given an assigment statement, x = expr:

$$wp(x = expr,\ O) = O\ [x \leftarrow expr]$$

i.e., replace all occurrences of x in O by expr.

Sequencing

Given a statement sequence: S1 ; S2 ;

$$wp(S1\ ;\ S2\ ;\ ,\ O) = wp(S1, wp(S2, O))$$

WP for Conditionals

wp(if (B) S1 else S2, O)

B && wp(S1, O) || not(B) && wp(S2, O)

B ➔ wp(S1, O) && not(B) ➔ wp(S2, O)

wp(if (B) S1, O)

B && wp(S1, O) || not(B) && O

B ➔ wp(S1, O) && not(B) ➔ O

WP for Sequence

{I}

S1;

S2;

{O}

{I}

S1;

{wp(S2,O)}

S2;

{O}

{I}

{wp(S1, wp(S2,O))}

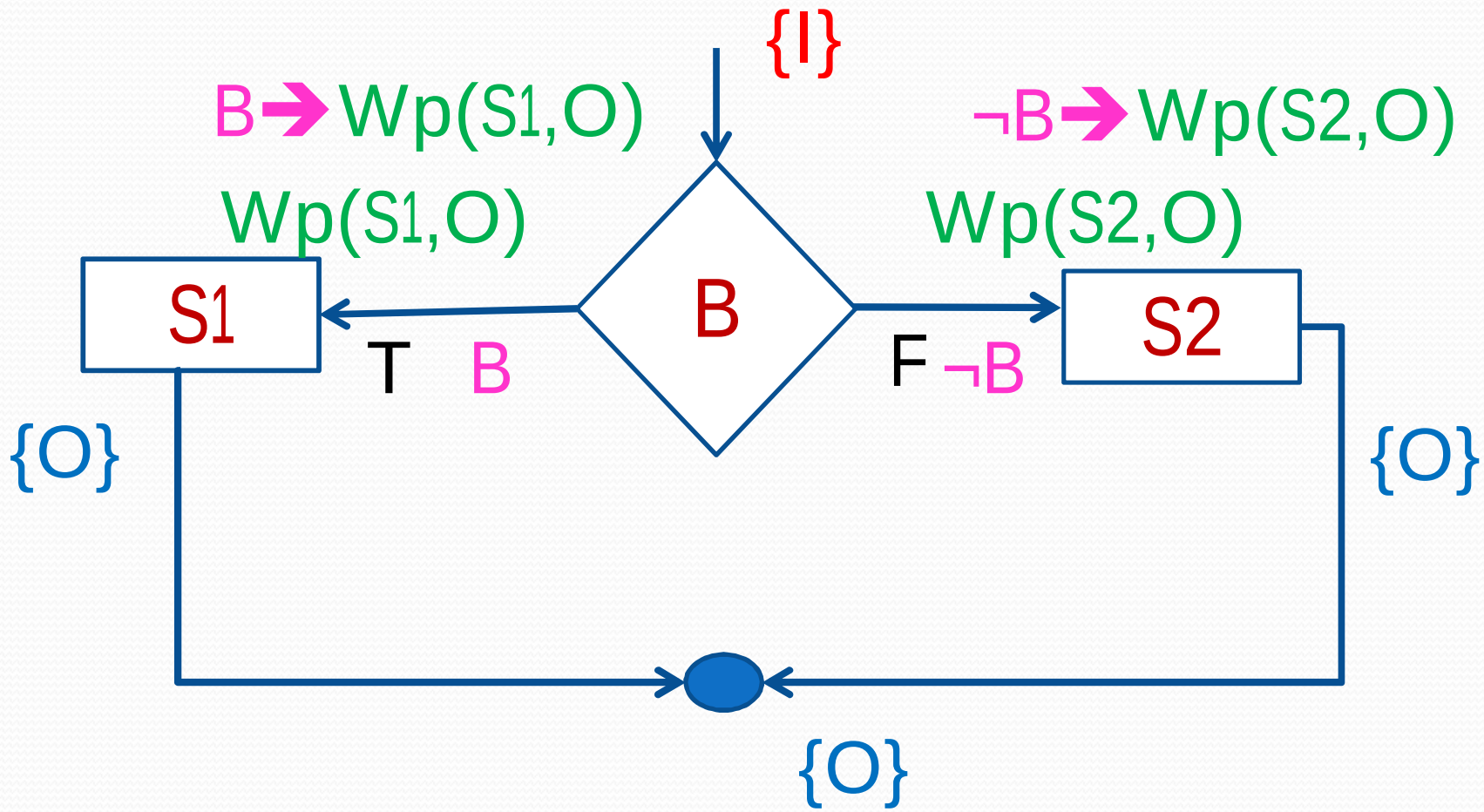
S1;

{wp(S2,O)}

S2;

{O}

WP of If -else



WP for Conditionals

If(B)
{B}
S1;
Else
{not B}
S2;
{O}

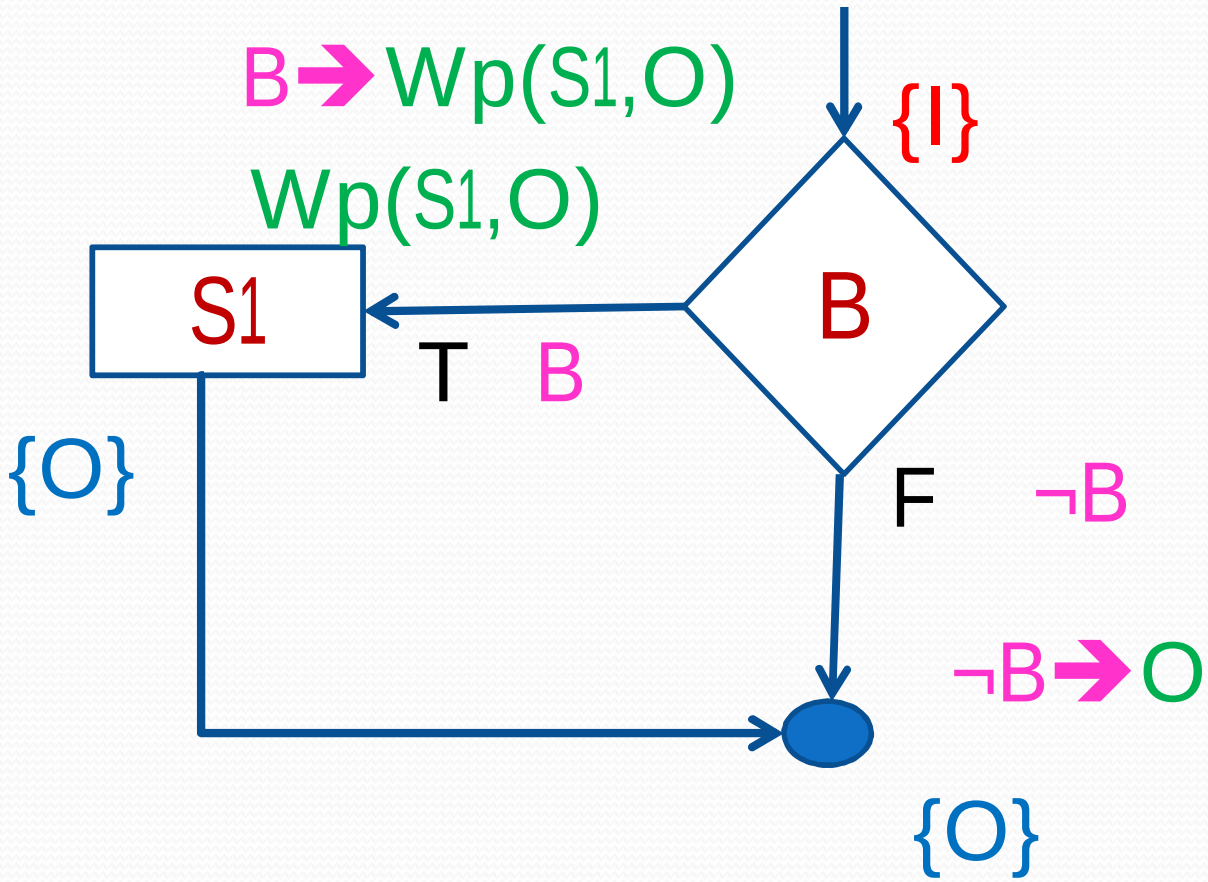
If(B)
{B}
S1; {O}
Else
{not B}
S2; {O}

If(B)
{B}
{wp(S1,O)} S1; {O}
Else
{not B}
{wp(S2,O)} S2; {O}

wp(if (B) S1 else S2, O)
B && wp(S1, O) || not(B) && wp(S2, O)

B ➔ wp(S1, O) && not(B) ➔ wp(S2, O)

WP of IF



WP for Conditionals

If(B)

{B}

S1;

Else

{not B}

{O}

If(B)

{B}

{wp(S1,O)} S1; {O}

Else

{not B} {O}

wp(if (B) S1, O)

B && wp(S1, O) || not(B) && O

B ➔ wp(S1, O) && not(B) ➔ O

Loop –infinite possible pre post

While loop

```
{I}
s = 1;
i = 1;
while (i < n) {
        i = i + 1;
        s = s + i;
    }
```

- {O}

If and goto code

```
{I}
s = 1;
i = 1;
Repeat:
if (i < n) {
        i = i + 1;
        s = s + i;
      goto Repeat;== if(i<n)...
    }
```

- {O}

wp(if (B) S1, O) :B ➔ wp(S1, O) && not(B) ➔ O

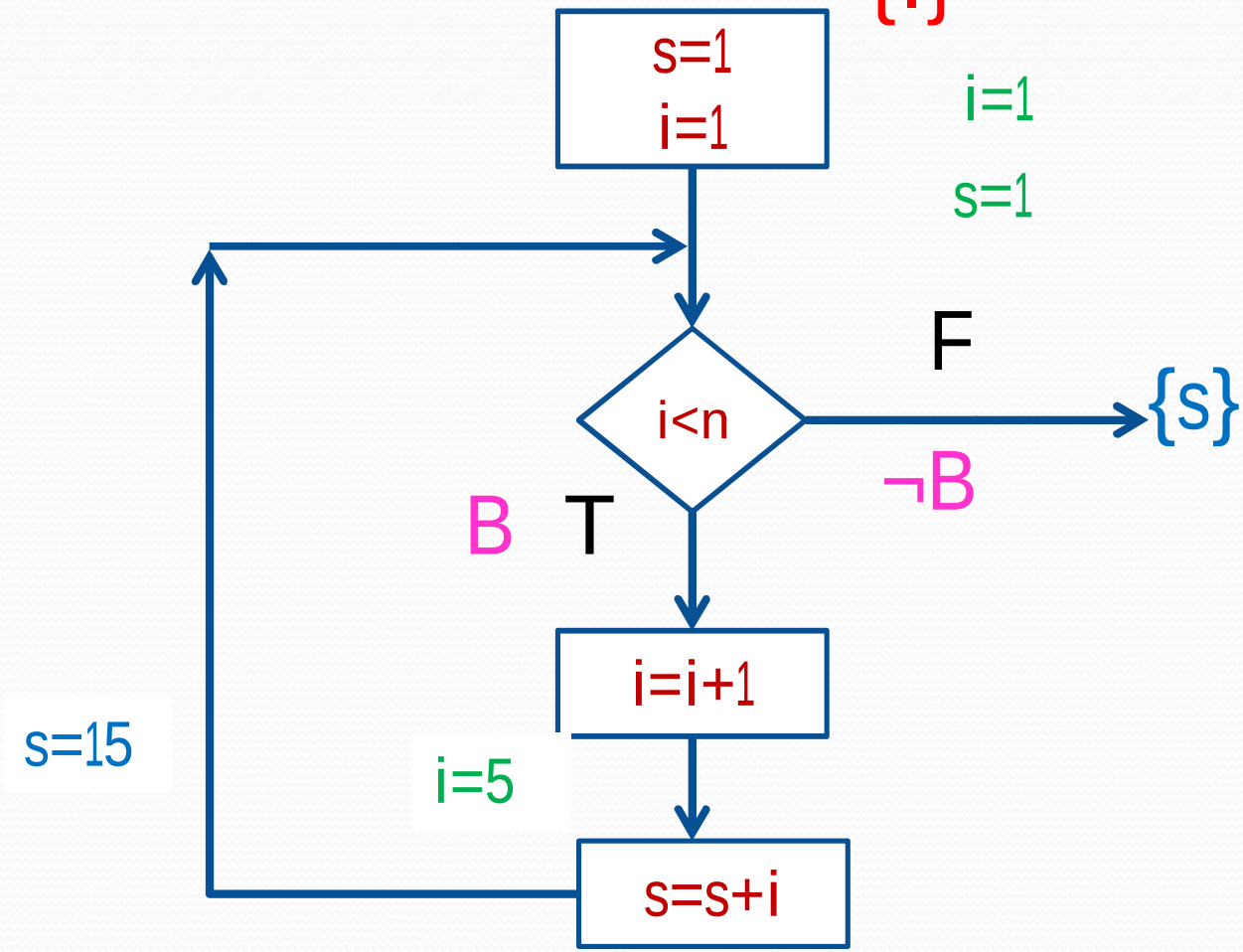
Separate If - WP

- Derive weakest precondition for the following: O{ x<6}
- {I}
- if (x >= 0) B1: (X>=0)
  x = x + 1; WP(S1,O): x+1<6

{P}

- if (x >= 1) B2: (x>=1)
- x = x + 2; WP(S2,O): x+2<6
- {O}
- If (B1) and (B2) : I ➔ wp(if1,P); P➔ wp(if2,O)
  - I➔wp(if1,wp(if2,O))
- If (B1) and not(B2): I➔wp(if1,O); P=O
- If not(B1) and (B2): I=P➔wp(if2,O)
- If not(B1) and not(b2): I=P=O

Sum of n numbers

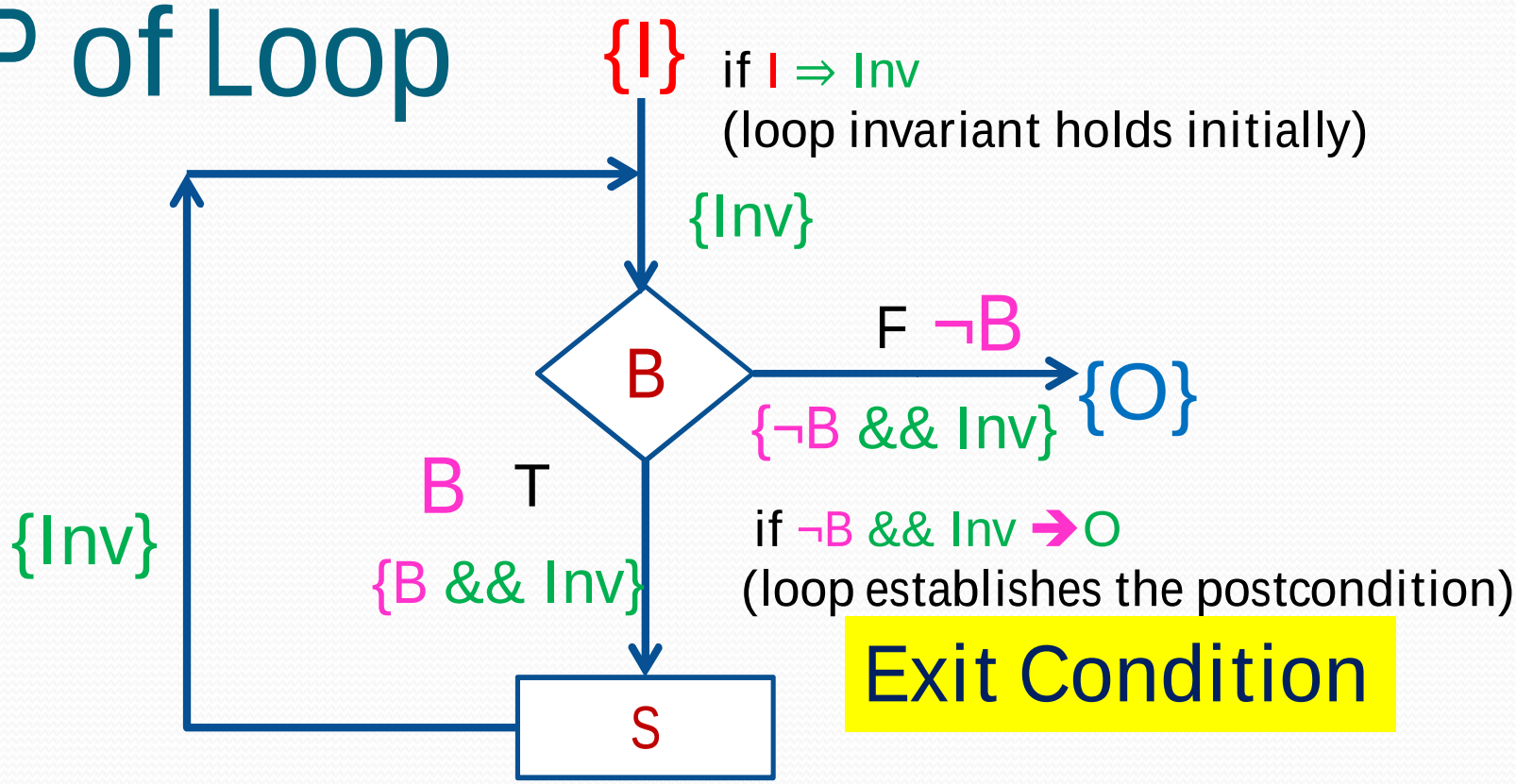


Single state of variables that satisfy the {O} irrespective of n/i?

Loop Invariant

- problem
  - we cannot *a priori* deduce how many times a loop iterates,
  - we cannot know how many times variables are modified
- A loop invariant is a property that must be true before and after each loop iteration. And more precisely, each time the condition of the loop is checked

WP of Loop



Exit Condition

{ Inv && B } **S** { Inv }
(loop invariant is preserved)

I ➔ wp(S, O)
Inv && B➔ wp(S, Inv)

Intra loop Condition

While Loop

Given a while statement, while (B) S, since S can be executed an unbounded number of times, wp cannot be derived simply from B, S, and O.

Hence, we define

$$wp(\text{while } (B)\ S,\ O) = Inv$$

where Inv is a **loop invariant** condition to be supplied by the user and satisfying:

(1) An intra-loop condition: Inv && B ==> wp(S, Inv)

(2) An exit condition: Inv && not(B) ==> O

Developing Loop Invariants

In practice, developing the right loop invariant is an iterative process of starting with an initial estimate and progressively refining it until the intra-loop and exit verification conditions are satisfied.

Tools can check VCs, but tools cannot* formulate the invariant in general – user must do this!

* - open research problem in the field!

Sum 1 to N

```
@requires n >= 1
@ensures   s = n*(n+1)/2
@program {
        s = 1;
        i = 1;
 @invariant        s = i*(i+1)/2
        while (i < n) {
            i = i + 1;
            s = s + i;
       }
        }
.
```

I && B ==> wp(S, I)

```
@requires n >= 1
@ensures s = n*(n+1)/2
program
 { s = 1;
   i = 1;
  @invariant
     s = i*(i+1)/2
   while (i < n) {
            i = i + 1;
            s = s + i;
    }
 }
```

s + (i+1) = (i+1)*(i+2)/2

s + i = i*(i+ 1)/2

s = i*(i+ 1)/2

Continuing: I && B ==> wp(S, I)

I: $s = i*(i+1)/2$

B: $i < n$

wp(S,I): $s + (i+1) = (i+1)*(i+2)/2$

$\equiv$ $s + (i+1) = (i^2 + 3*i + 2)/2$

$\equiv$ $2*s + 2*(i+1) = i^2 + 3*i + 2$

$\equiv$ $s = (i^2 + i)/2 = i*(i+1)/2$

I && not(B) ==> O

O: s = n * (n + 1) / 2

I: s = i * (i + 1) / 2

not(B): i >= n

I && not(B): s = i * (i + 1) / 2 && i >= n

Unfortunately, I && not(B) ==> O is not valid!

Question: How can we "strengthen" I so that the above VC holds?

Answer: I: s = i * (i + 1) / 2 && i =< n

Total Correctness

```
@requires  n >= 1
@ensures  s = n*(n+1)/2
@program { i = 1;
           s = 1;
           @invariant ...
           while (i =< n) {
              i = i + 1;
              s = s + i;
           }
        }
```

We need to formulate a non-negative integer-valued function (over a subset of program variables) whose value strictly decreases for each iteration of the loop.

Consider the function:

$$g(i, n, s) = n - i$$

Total Correctness (cont'd)

```
@requires  n >= 1
@ensures   s = n*(n+1)/2
@program { i = 1;
           s = 1;
           @invariant ...
           while (i =< n) {
              i = i + 1;
              s = s + i;
           }
         }
```

Consider

g(i, n, s) = n – i

Start of Loop:

n >= 1 && i = 1

=> g(i, n, s) >= 0

Iteration in Loop:

i =< n

==> g(i, n, s) >= 0

Each iteration strictly decreases the value of g(i, n, s).
Hence, execution cannot continue in the loop indefinitely.

Exercise

- Derive the weakest precondition for sum of squares(use loop)
- Derive the weakest precondition for factorial of a number(use loop)