19CSE205 Program Reasoning Weakest Precondition

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:

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

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).

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

Assignment

Given an assigment statement, x = expr:

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

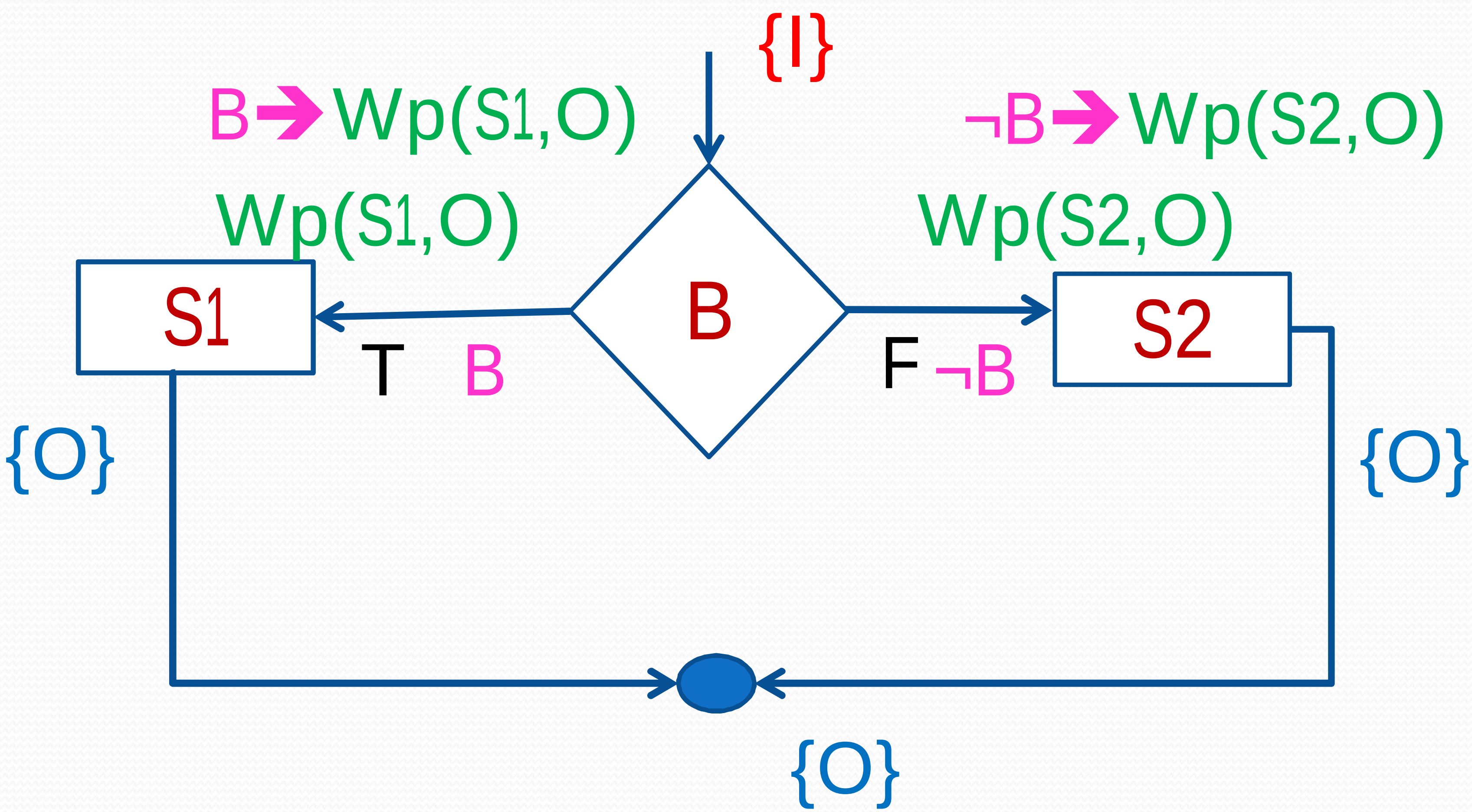
i.e., replace all occurrences of x in O by expr.

Sequencing

Given a statement sequence: S1 ; S2 ;

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

WP of If -else



Conditional Statements

Statement: if (B) S1 else S2

How to define: wp(if (B) S1 else S2, O)

If-Part (IP): _wp(S1, O)_

Else-Part (EP): _wp(S2, O)_

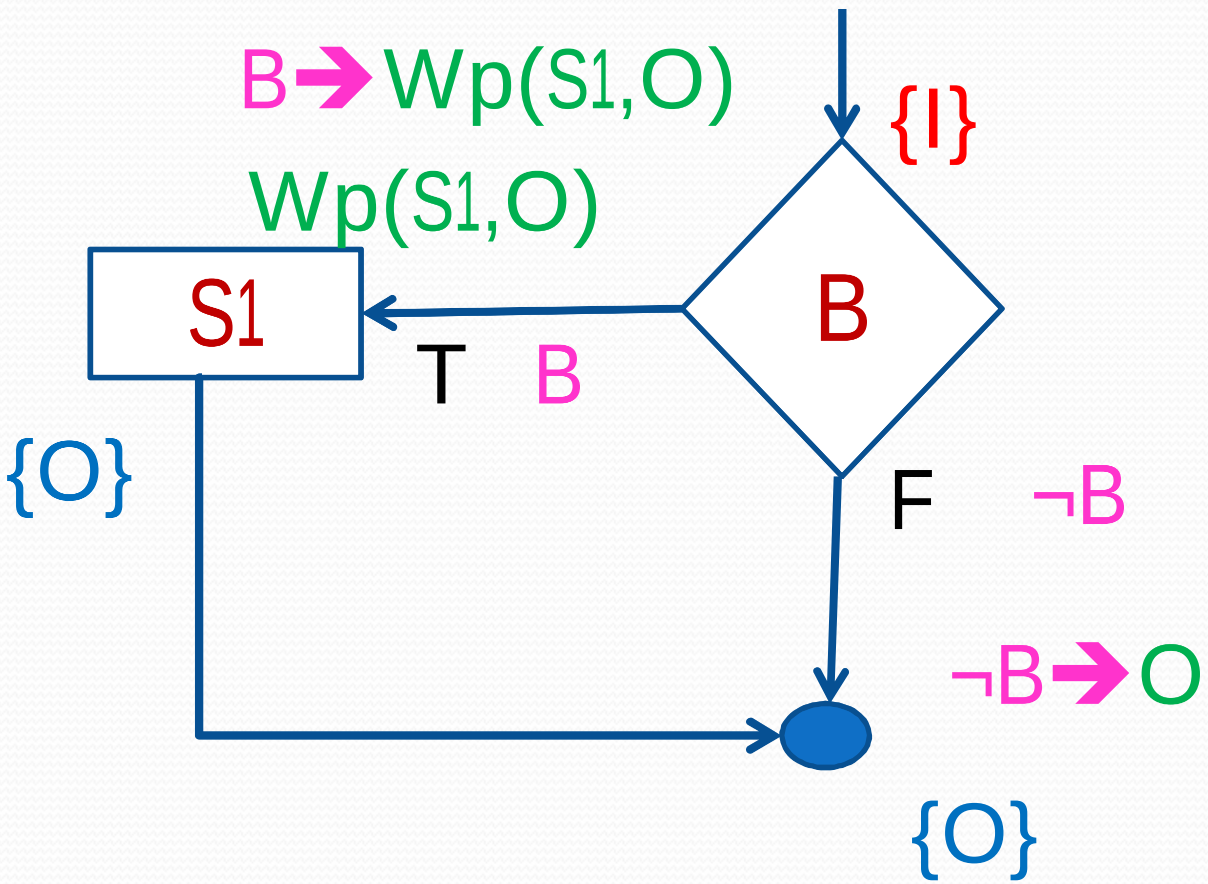
wp(if (B) S1 else S2, O) =

B ==> wp(S1,O)

&&

not(B) ==> wp(S2,O)

WP of IF



Conditional Statements

Given the statement: if (B) S1

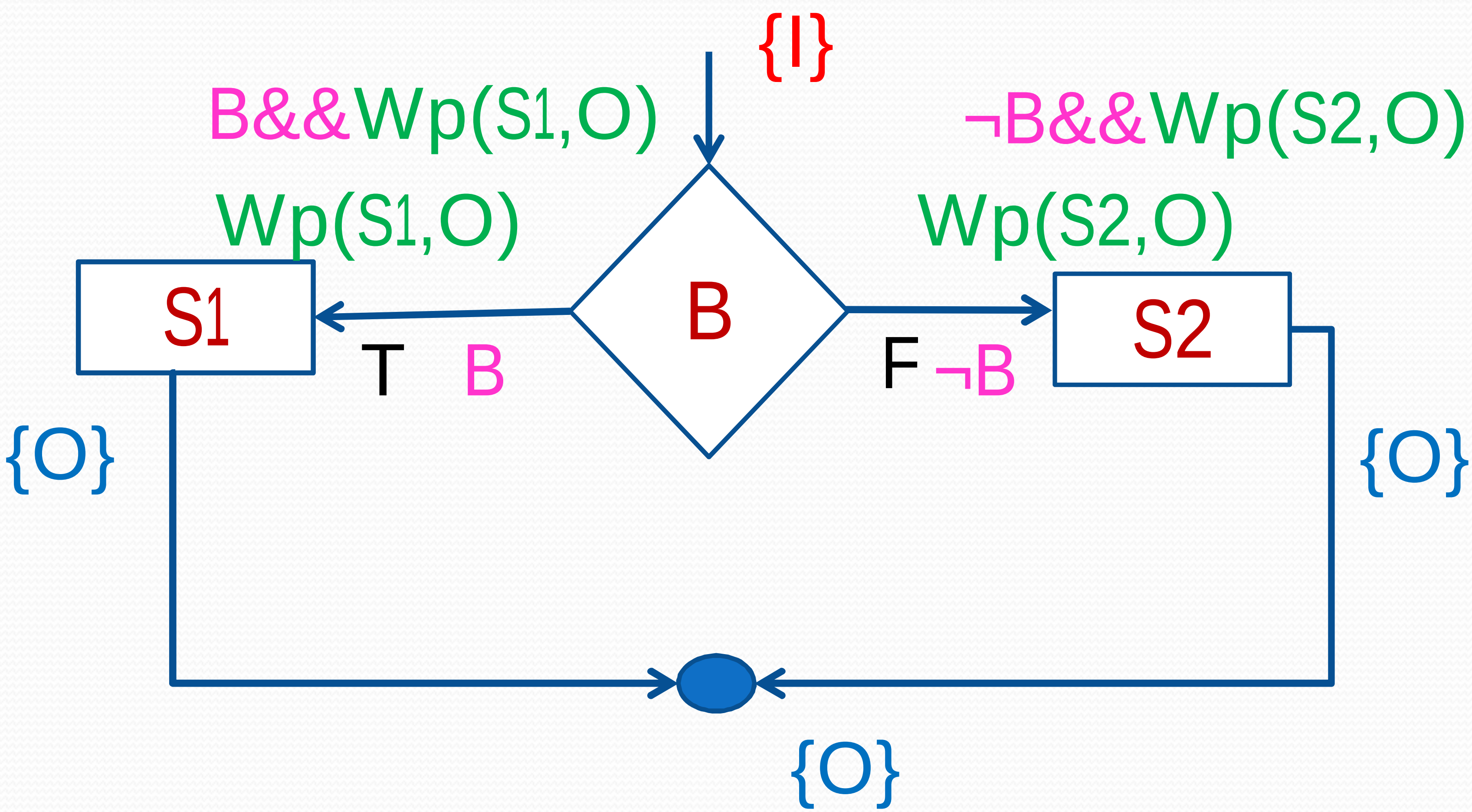
How should we define:

wp(if (B) S1, O) = B ==> wp(S1,O)

&&

__not(B) ==> O___

WP of If else –Another method



Conditional Statements

Statement: if (B) S1 else S2

How to define: wp(if (B) S1 else S2, O)

If-Part (IP): _wp(S1, O)_

Else-Part (EP): _wp(S2, O)_

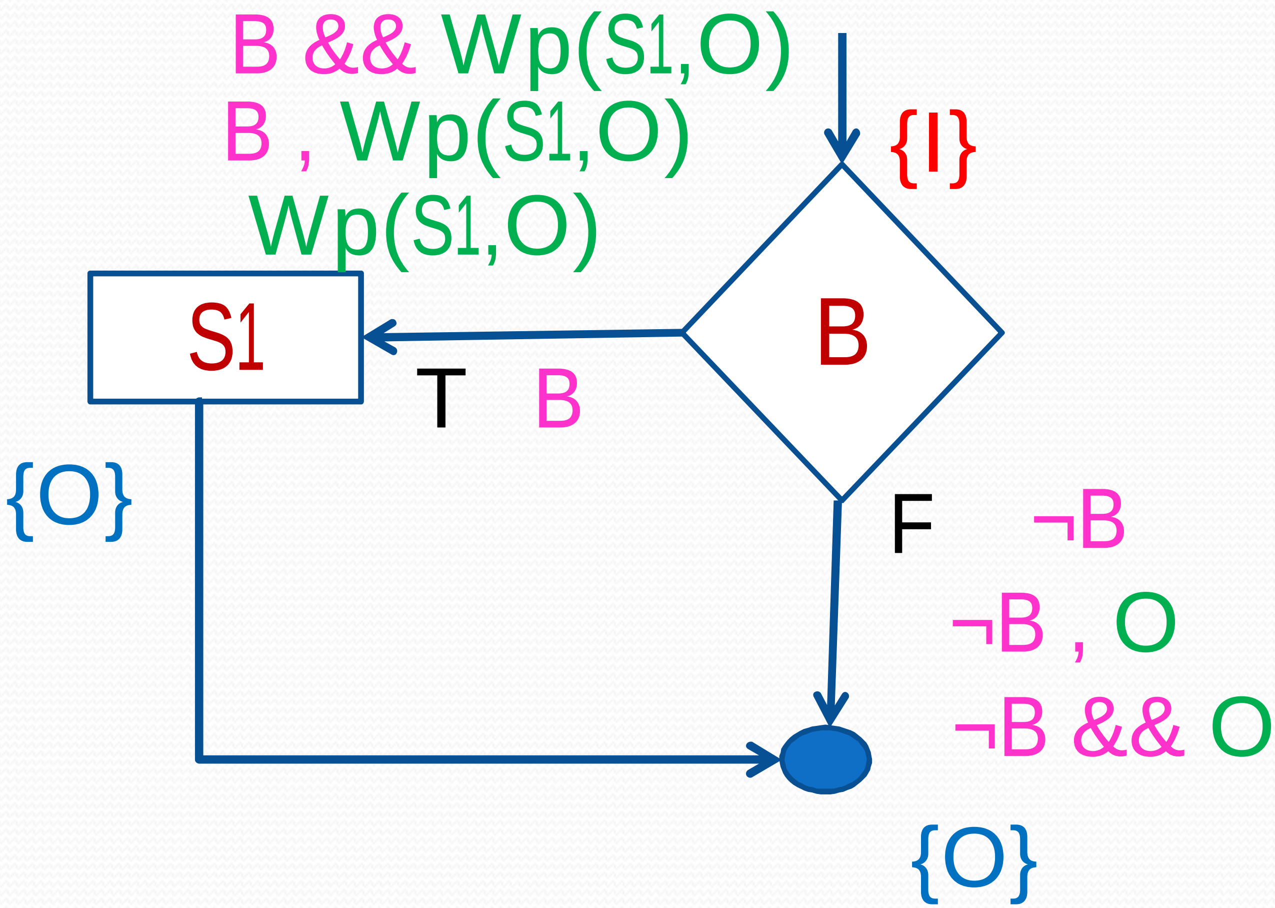
wp(if (B) S1 else S2, O) =

B && wp(S1,O)

||

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

WP of If - Another method



Conditional Statements

Given the statement: if (B) S1

How should we define:

wp(if (B) S1, O) = B && wp(S1,O)

||

__not(B) && O___

WP for Conditionals

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

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

Checking Equivalence

We can check equivalence between:

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

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

Abbreviate: wp(S1, O) ➔ P wp(S2, O) ➔ Q

Using Alt-Ergo, we can quickly check equivalence between:

B && P || not(B) && Q and

(B ==>P) && (not(B) ==> Q)

Important Observation

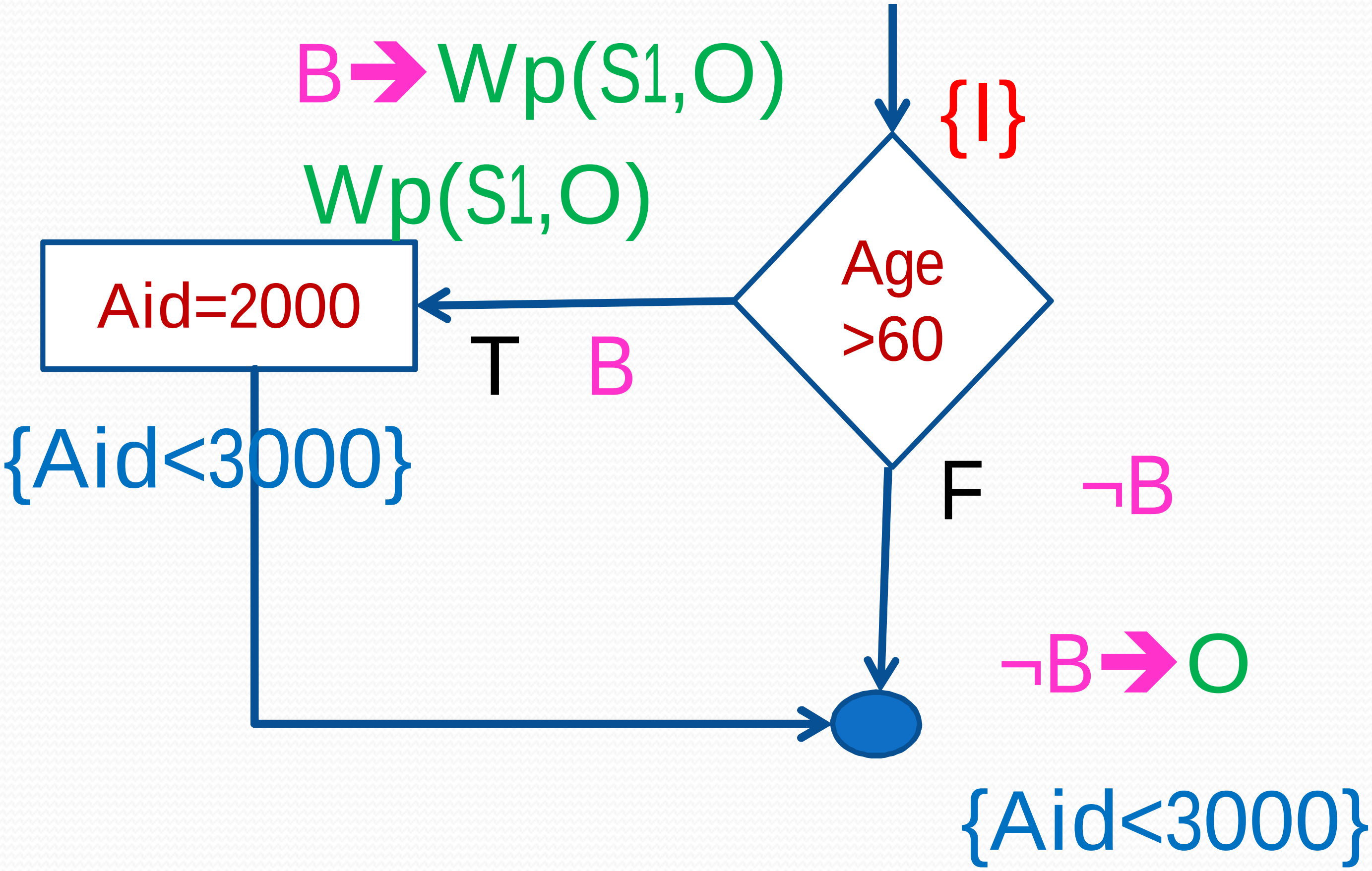
Programs that are easy to verify are also programs that are easy to understand.

Easy to Verify ==> Easy to Understand

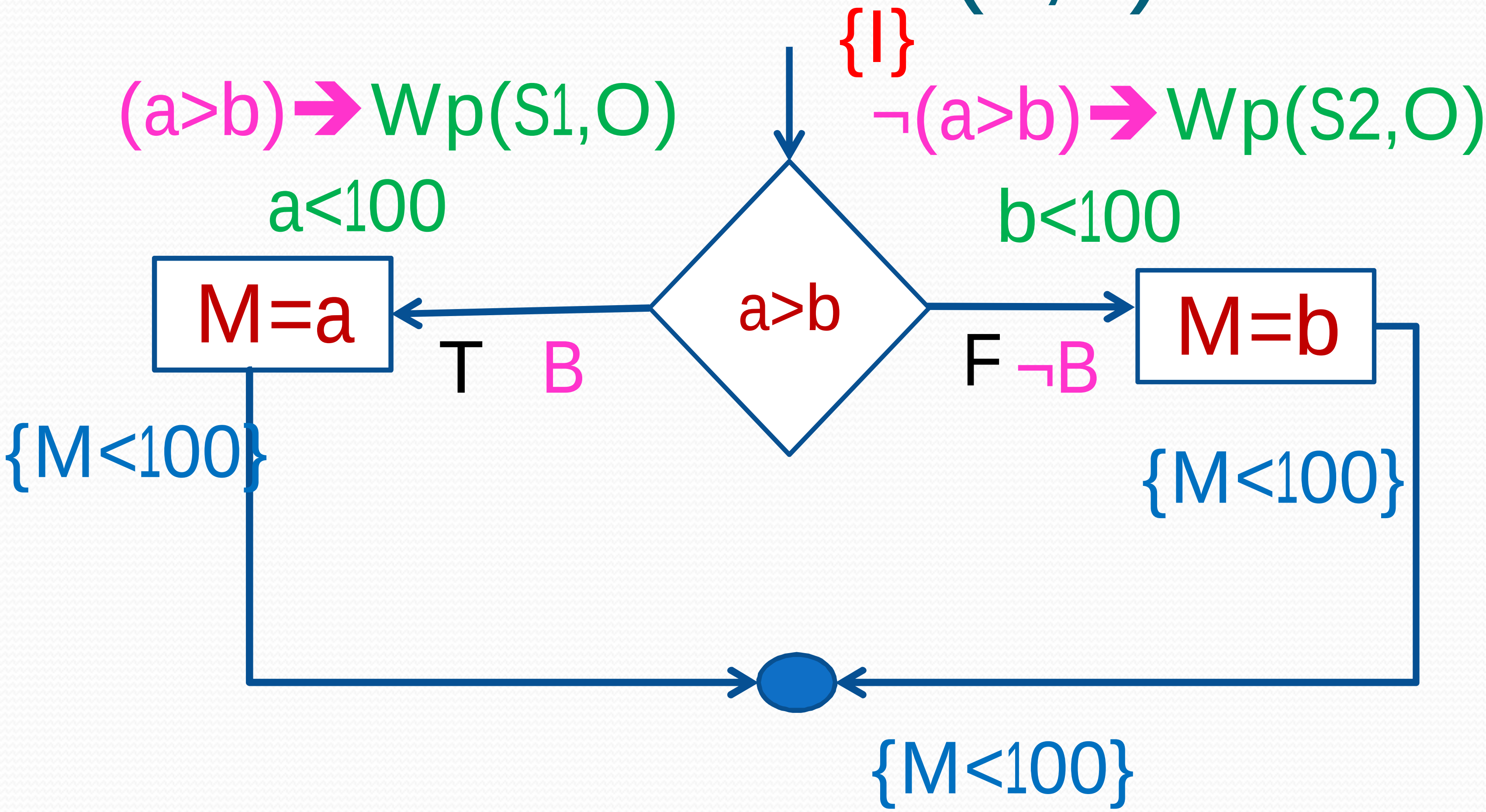
&&

Easy to Understand ==> Easy to Verify

WP of IF



WP of If –else-max(a,b)



Max Example

- Derive the weakest precondition for the following Code Snippet :

```
if(a==b)
S1:   b=2*a+1;
else
S2:  b=2*a;
O: {b>1}
```

- B: (a==b)
- wp(s1,O):2*a+1>1 =a>0
- wp(s2,O): 2*a>1

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

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

Example

- Derive the weakest precondition for the following Code Snippet :

```
if(a==b)
S1:   b=2*a+1;
else
S2:  b=2*a;
O: {b>1}
```

- B: (a==b)
- wp(s1,O):2*a+1>1 =a>0
- wp(s2,O): 2*a>1

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

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

Nested if Vs individual if

- Derive weakest precondition for the following: O{ x<6}
- if (x >= 0)
  - x = x + 1;
- else if (x >= 1)
  - x = x + 2;

if (x >= 0)
    x = x + 1;
if (x >= 1)
    x = x + 2;

- Exercise:
  - Based on x, y coordinates assign the quadrant

Nested If- Weakest precondition

- Derive weakest precondition for the following: O{ x<6}
- if (x >= 0) B1: (X>=0)

  x = x + 1; WP(S1,O): x+1<6
- else if (x >= 1) not(B1): B2: (x>=1)
- x = x + 2; WP(S2,O): x+2<6
- (B1➔wp(S1,O))&&(not(B1)➔wp(if (B2) S2,O))
- (B1➔wp(S1,O))&&(not(B1)➔((B2➔wp(S2,O))&&

  (not(B2)➔O)))

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, 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

Continued

- If (B1) and (B2) :I➔wp(if1,wp(if2,O))
- wp(if (B1) S1, wp(if (B2) S2,O))
- wp(if (B1) S1,((B2➔wp(S2,O))&& (not(B2)➔O)))
- B1➔wp(S1, ((B2➔wp(S2,O))&& (not(B2)➔O))) &&

  Not(B1)➔((B2➔wp(S2,O))&& (not(B2)➔O))
- (x>=0)➔wp(x=x+1, ((x>=1)➔wp(x=x+2,x<6))&& (not(x>=1)➔(x<6)))) && Not(x>=0)➔ ((x>=1)➔wp(x=x+2,x<6))&& (not(x>=1)➔(x<6))

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

Continued

- (x>=0)➔wp(x=x+1, ((x>=1)➔wp(x=x+2,x<6))&& (not(x>=1)➔(x<6)))) && Not(x>=0)➔ ((x>=1)➔wp(x=x+2,x<6))&& (not(x>=1)➔(x<6))
- (x>=0)➔wp(x=x+1, ((x>=1)➔(x+2<6)&& (not(x>=1)➔(x<6)))) && Not(x>=0)➔ ((x>=1)➔(x+2<6)&& (not(x>=1)➔(x<6))
- (x>=0)➔(x+1>=1)➔(x+1+2<6)&& (not(x+1>=1)➔(x+1<6)) && Not(x>=0)➔ ((x>=1)➔(x+2<6)&& (not(x>=1)➔(x<6))

continued

- If (B1) and not(B2): I➔wp(if1,O);
- wp(if (B1) S1,O)
- B1➔wp(S1, O) && Not(B1)➔O
- (x>=0) ➔wp(x=x+1, x<6) && Not (x>=0) ➔(x<6)
- (x>=0) ➔(x+1<6) && Not (x>=0) ➔(x<6)
- If not(B1) and (B2): I=P➔wp(if2,O)
- B2➔wp(S2, O) && Not(B2)➔O
- (x>=1) ➔wp(x=x+2, x<6) && Not (x>=1) ➔(x<6)
- (x>=1) ➔(x+2<6) && Not (x>=1) ➔(x<6)
- If not(B1) and not(b2): I=P=O
- I➔(x<6)

(B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

Alt-Ergo for int vs real values

```
goal g_1 :
  forall x,y,z,t:int.
    0 <= y + z <= 1  ->
    x + t + y + z = 1 ->
    y + z <> 0 ->
    x + t = 0
```

```
# [answer] Valid (0.0720 seconds) (5 steps)
```

```
goal g_1 :
  forall x,y,z,t: real.
    0.0 <= y + z <= 1.0  ->
    x + t + y + z = 1.0 ->
    y + z <> 0.0 ->
    x + t = 0.0
```

```
# [answer] unknown (0.0970 seconds) (6 steps)
```

Exercise

```
a. x = 25.0;
if (y != (x - 10.0))
        x = x - 10.0;
else
        x = x / 2.0;

c. if (y < 15.0 && y >= 0.0)
        x = 5 * y;
else
        x = 2 * y;
```

```
b. if (y < 15.0)
        if (y >= 0.0)
                x = 5 * y;
        else
                x = 2 * y;
else
        x = 3 * y;

d. if (x > y) {
        temp = x;
        x = y;
        y = temp;}
```

Compare Two Simple Programs

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

Program 1

```
@requires marks = 75
@ensures  grade = B
@program {
  if (marks > 90)
        grade = A;
  else  if (marks > 70)
            grade = B;
        else if (marks > 50)
                  grade = C;
             else grade = F;
}
.
```

Program2

Verification Conditions

In general, if Program 1 had n cases, the size of the Verification Condition generated is $O(2^n)$.

In general, if Program 2 had n cases, the size of the Verification Condition generated is O(n).

Program 1 would have an exponential number of control paths, in general. Also, the variable grade is repeatedly modified, and this also contributes to greater code complexity. Program 2 assigns grade exactly once, hence is less complex.