

# More Inference Rules

Curt Clifton

Rose-Hulman Institute of Technology

Hmm,  
elephants

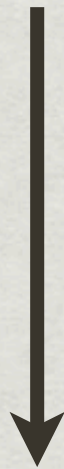


# Last Time

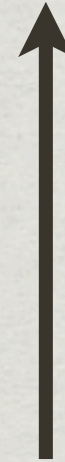
- \* Assignment
- \* Sequencing (a.k.a., composition)

# Assignment Rule

WHATEVER IS  
TRUE ABOUT  $e$   
BEFORE IS TRUE  
ABOUT  $v$  AFTER



```
//@ assert  $P(e)$ ;  
 $v = e$ ;  
//@ assert  $P(v)$ ;
```



WHAT MUST BE  
TRUE ABOUT  $e$   
BEFORE IF WE WANT  
SOME PROPERTY TO  
BE TRUE ABOUT  $v$

# Composition Rule

IF:

**//@ assert P<sub>1</sub>;**

**S<sub>1</sub>;**

**//@ assert Q<sub>1</sub>;**

AND

**//@ assert P<sub>2</sub>;**

**S<sub>2</sub>;**

**//@ assert Q<sub>2</sub>;**

AND

**Q<sub>1</sub> ==> P<sub>2</sub>**

THEN:

**//@ assert P<sub>1</sub>;**

**S<sub>1</sub>;**

**//@ assert Q<sub>1</sub>;**

**//@ assert P<sub>2</sub>;**

**S<sub>2</sub>;**

**//@ assert Q<sub>2</sub>;**

IMPORTANT: IMPLICATIONS GO DOWN

# More Bites

- \* If-then-else
- \* If-then
- \* While

# If-Then-Else Rule

**IF:**

**//@ assert  $P_1$  && B;**

**$S_1$ ;**

**//@ assert Q;**

**AND**

**//@ assert  $P_2$  && !B;**

**$S_2$ ;**

**//@ assert Q;**

**AND**

**$P$  && B  $\implies P_1$**

**AND**

**$P$  && !B  $\implies P_2$**

**THEN:**

**//@ assert P;**

**if (B) {**

**$S_1$ ;**

**} else {**

**$S_2$ ;**

**}**

**//@ assert Q;**

**USEFUL WHEN WORKING BACKWARDS:  
IN GENERAL P IS  $(B \implies P_1) \&\& (!B \implies P_2)$**

# Another Bite of Elephant—What's P?

```
    //@ assert P ;  
if (x > 0) {  
    y = x;  
} else {  
    y = -x;  
}  
  
    //@ assert y >= 0;
```

# What's P?

```
    //@ assert P;
if (x > 0) {
5:    //@ assert P && x > 0; // from above
4:    //@ assert x >= 0; // P1
    y = x;
2:    //@ assert y >= 0;
} else {
6:    //@ assert P && x <= 0; // from above
3:    //@ assert -x >= 0; // P2
    y = -x;
1:    //@ assert y >= 0;
}
    //@ assert y >= 0;
```

**NEED:**

$((P \ \&\& \ x \ > \ 0) \implies x \geq 0) \ \&\&$   
 $((P \ \&\& \ x \leq 0) \implies -x \geq 0)$

**WEAKEST PRECONDITION: P IS true**

# If-Then-Else Rule

**IF:**

**//@ assert P<sub>1</sub> && B;**

**S<sub>1</sub>;**

**//@ assert Q;**

**AND**

**//@ assert P<sub>2</sub> && !B;**

**S<sub>2</sub>;**

**//@ assert Q;**

**AND**

**P && B ==> P<sub>1</sub>**

**AND**

**P && !B ==> P<sub>2</sub>**

**THEN:**

**//@ assert P;**

**if (B) {**

**S<sub>1</sub>;**

**} else {**

**S<sub>2</sub>;**

**}**

**//@ assert Q;**

**IN GENERAL P IS**

**(B ==> P<sub>1</sub>) && (!B ==> P<sub>2</sub>)**

# Another Example

FIND THE WEAKEST PRE-CONDITION, P

```
//@ assert P;  
if (i % 2 != 0) {  
    i = i / 2;  
} else {  
    i = i + 1;  
}  
  
//@ assert i >= 0;
```

**RECALL  $A \implies B$  IS EQUIVALENT TO  $B \vee \neg A$**

# Another Example

P

```
10: //@ assert i >= 0;
9: //@ assert (i >= 0 || i%2 == 0) && (i >= -1 || i%2 != 0);
// ==> by arithmetic
8: //@ assert (i/2 >= 0 || i%2 == 0) && (i+1 >= 0 || i%2 != 0);
// ==> by defn. of implication
7: //@ assert (i%2 != 0 ==> i/2 >= 0) && (i%2 == 0 ==> i+1 >= 0);
if (i % 2 != 0) {
5: //@ assert P && i % 2 != 0;
4: //@ assert i / 2 >= 0; // <----- P1
i = i / 2;
2: //@ assert i >= 0;
} else {
6: //@ assert P && i % 2 == 0;
3: //@ assert i + 1 >= 0; // <----- P2
i = i + 1;
1: //@ assert i >= 0;
}
//@ assert i >= 0;
```

$(B \implies P_1) \ \&\& \ (!B \implies P_2)$

# If-Then Rule

**IF:**

```
    //@ assert P1;  
S;  
    //@ assert Q;
```

**WHERE**

```
P && B ==> P1 AND  
P && !B ==> Q
```

**THEN:**

```
    //@ assert P;  
if (B) {  
    S;  
}  
    //@ assert Q;
```

**USEFUL WHEN WORKING BACKWARDS:  
IN GENERAL P IS (B ==> P<sub>1</sub>) && (!B ==> Q)**

# Example:

## Prove Correctness

```
//@ assert true;  
if (x % 2 != 0) {  
    x = x - 1;  
}  
  
//@ assert x % 2 == 0;
```

# Example:

## Prove Correctness

```
    //@ assert true;
if (x % 2 != 0) {
2:    //@ assert true && x % 2 != 0;
4:    //@ assert (x - 1) % 2 == 0;
3:    x' = x - 1; // tick trick
1:    //@ assert x' % 2 == 0;
}
    //@ assert x % 2 == 0;
```

# Quote of the Day

**EVERY PROGRAM HAS AT LEAST ONE BUG AND CAN BE SHORTENED BY AT LEAST ONE INSTRUCTION — FROM WHICH, BY INDUCTION, IT IS EVIDENT THAT EVERY PROGRAM CAN BE REDUCED TO ONE INSTRUCTION THAT DOES NOT WORK**

**– KEN ARNOLD**

# While Rule

- \* P is:
  - \* true for every turn of the loop
  - \* called the “loop invariant”
  - \* true before the loop, true after each iteration, true after the statement

**P SHOULD BE TRUE WHENEVER THE LOOP CONDITION IS EVALUATED**

```
IF:
    //@ assert P && B;
S;
    //@ assert P;
AND
P && !B ==> Q
```

```
THEN:
    //@ assert P;
while (B) {
    //@ assert P && B;
S;
    //@ assert P;
}
    //@ assert P && !B;
    //@ assert Q;
```

# Prove Correctness

```
//@ assert true;  
x = 0;  
while (x < 20) {  
    x = x + 2;  
}  
//@ assert x == 20;
```

# Bad Guess

**GUESS LOOP INVARIANT:  $x \leq 20$**

```
    //@ assert true;
4:    //@ assert 0 <= 20;
    x = 0;
1:    //@ assert x <= 20;
    while (x < 20) {
3:        //@ assert x <= 20 && x < 20;
7:        ??? can't make this leap, bad loop invariant guess
6:        //@ assert x + 2 <= 20;
5:        x' = x + 2; // tick-trick
2:        //@ assert x' <= 20;
    }
    //@ assert x == 20;
```

# Better Guess

**LOOP INVARIANT:  $x \leq 20 \ \&\& \ x \% 2 == 0$**

```
    //@ assert true;
4:    //@ assert 0 <= 20 && 0 % 2 == 0;
    x = 0;
1:    //@ assert x <= 20 && x % 2 == 0;
    while (x < 20) {
3:        //@ assert x <= 20 && x % 2 == 0 && x < 20;
7:        //@ assert x <= 18 && x % 2 == 0;
6:        //@ assert x + 2 <= 20 && (x + 2) % 2 == 0;
5:        x' = x + 2; // tick-trick
2:        //@ assert x' <= 20 && x' % 2 == 0;
    }
8:    //@ assert x <= 20 && x % 2 == 0 && x >= 20;
    //@ assert x == 20;
```



**IMPLICATIONS  
HOLD READING  
DOWN THE  
PAGE**