What Does this Program Do?

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

How can you be sure?

Answer

- Integer division
  - Compute quotient **Q** and remainder **R** of **X** divided by **Y**, for non-negative integer **X** and positive integer **Y**
  - Expressed as a function returning two results
    <Q, R> ← DIVIDE(X, Y)
  - Expressed as a relation of four variables
    DIVIDE(X, Y, Q, R)

Preconditions

- What must be true about the inputs to this program in order for the program to successfully execute?

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

Preconditions

- $X \geq 0 \wedge Y > 0$
  - The value of X is non-negative and
  - The value of Y is positive

```
Q ← 0
R ← X
while (R ≥ Y)
    R ← R − Y
    Q ← Q + 1
```

Postconditions

- What must be true about the program output variables after the program has completed execution?
  - Expressed in terms of input and output variables
  - Assuming that the program terminates

```
Q ← 0
R ← X
while (R ≥ Y)
    R ← R − Y
    Q ← Q + 1
```

Postconditions

- $Y > 0 \wedge$
- $X \geq 0$

```
Q ← 0
R ← X
while (R ≥ Y)
    R ← R − Y
    Q ← Q + 1
```

Postconditions

- **$Y > 0 \wedge$**
- **$X \geq 0 \wedge$**
- **$Q \geq 0$**

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

Postconditions

- **$Y > R \geq 0 \wedge$**
- **$X \geq 0 \wedge$**
- **$Q \geq 0$**

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

Postconditions

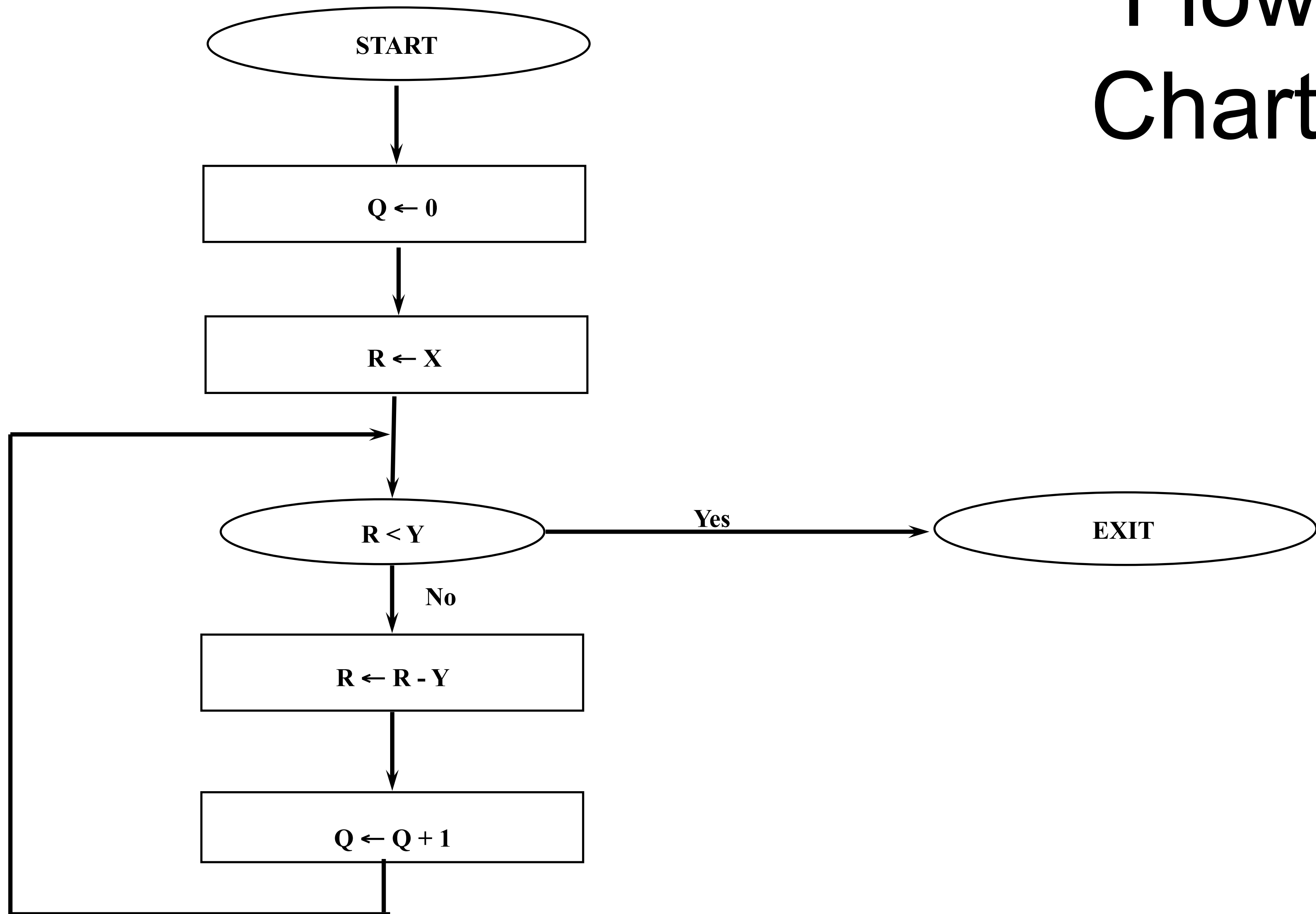
- **$Y > R \geq 0 \land$**
- **$X \geq 0 \land$**
- **$Q \geq 0 \land$**
- **$X = Q * Y + R$**

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

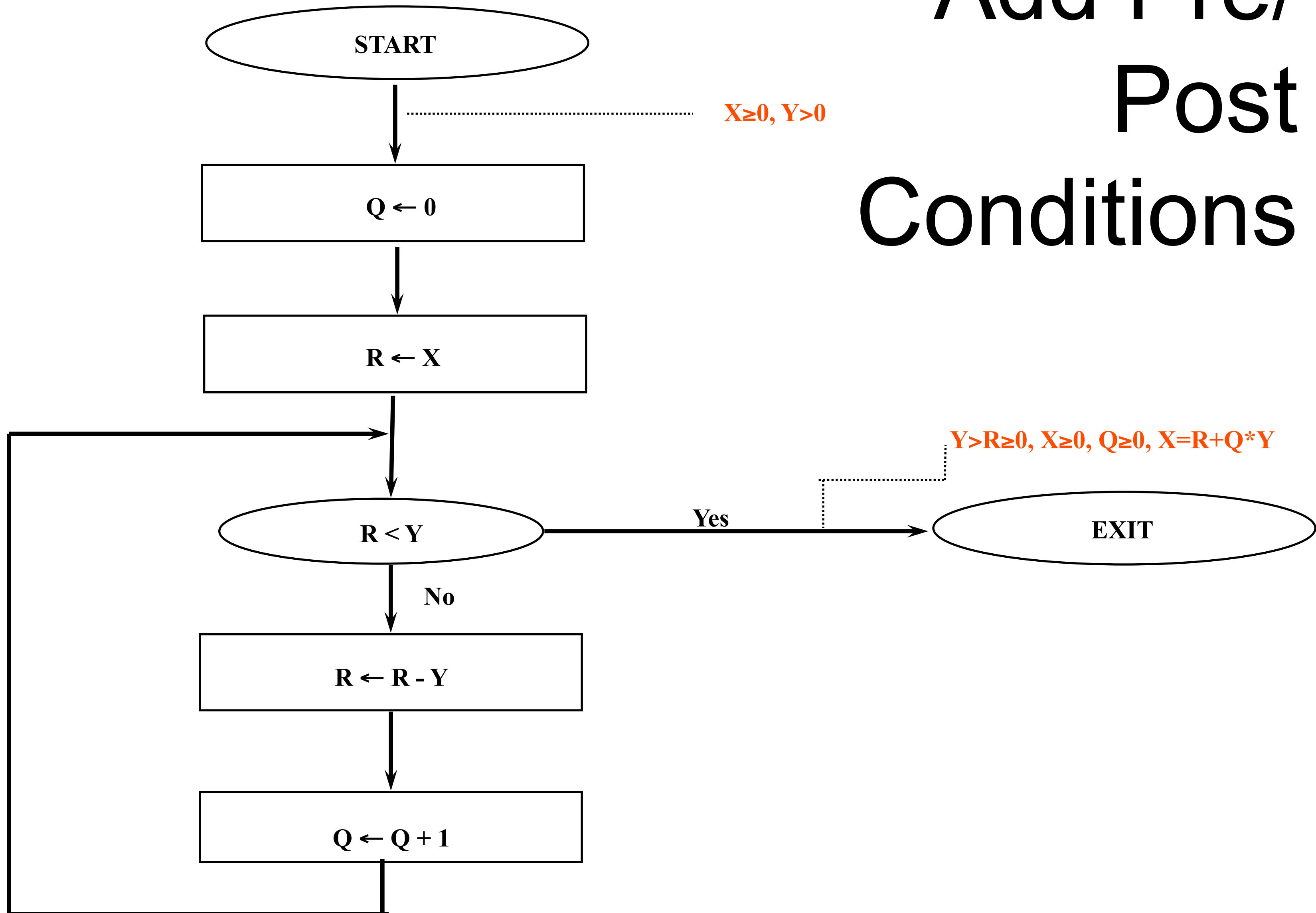
Proof Plan

- Construct flow chart
- Annotate with preconditions
- Add loop invariants at intermediate program points based on the type of statement executed
  - Assignment
  - Conditional
  - Loop

Flow Chart



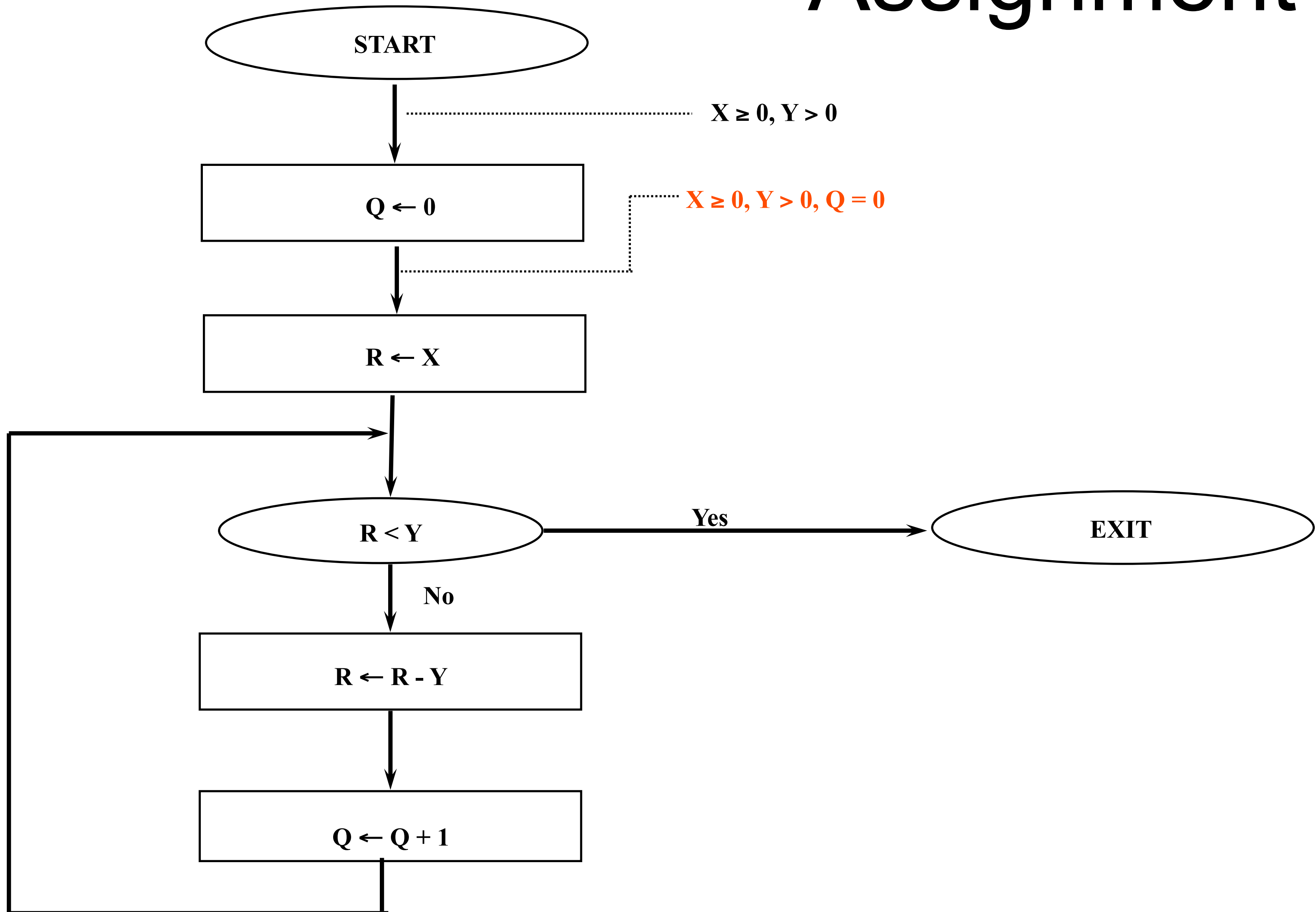
Add Pre/ Post Conditions



Simple Assignment

- The statement **{Q ← 0}** guarantees that **Q** has the value **0** after it executes
- Moreover, all other variables remain unchanged
  - Assuming no variable aliasing
- Therefore, the invariant after the assignment execution looks like **$X \geq 0, Y > 0, Q = 0$**

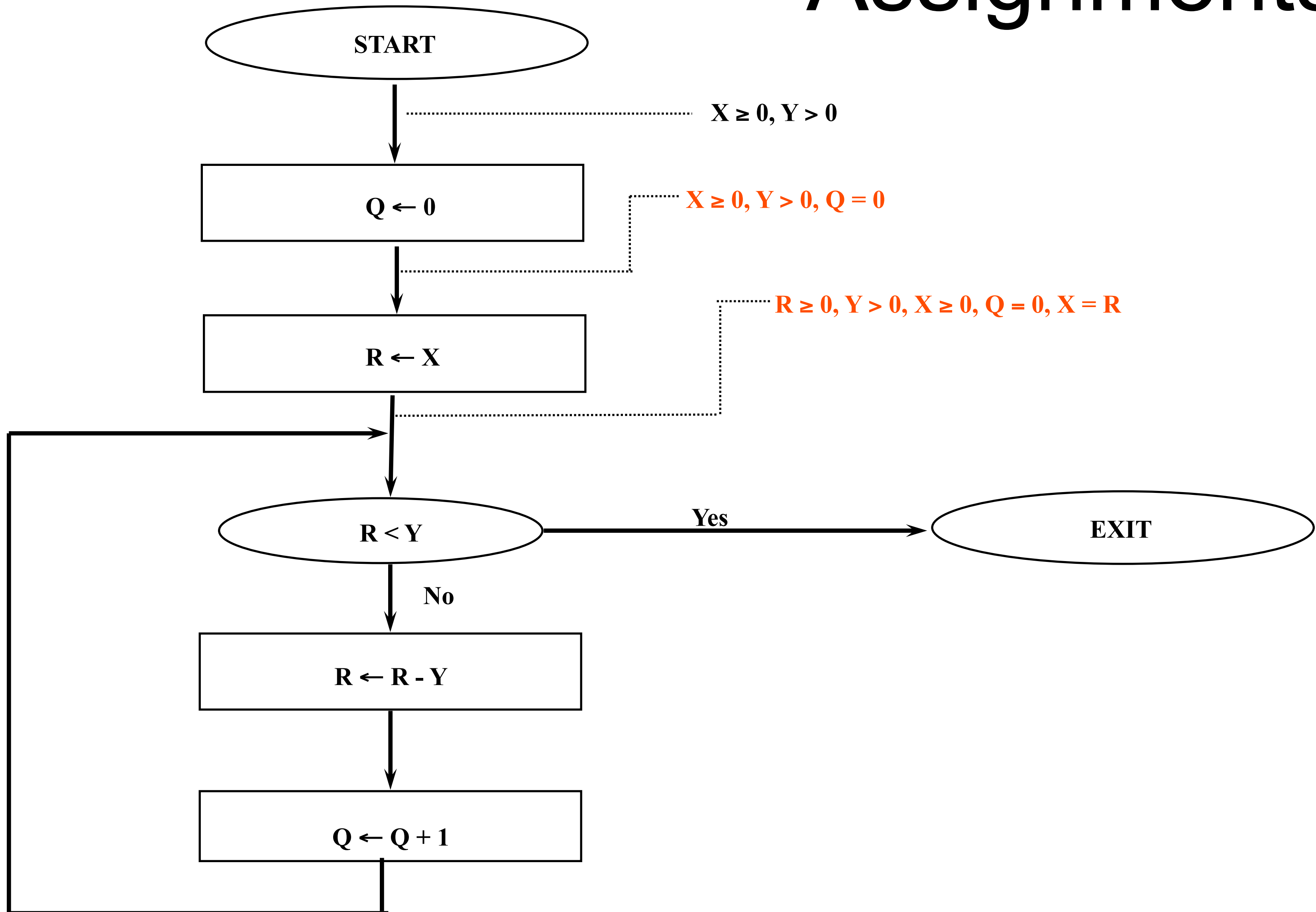
Assignment



Slightly More Complex Assignment

- The statement **{R ← X}** guarantees that **R** has the same value as **X** after it executes
  - So, **X ≥ 0, Y > 0, Q = 0** becomes **X ≥ 0, Y > 0, Q = 0, R = X**
- We can also use the laws of algebra to manipulate other assertions
  - The assertion is algebraically equivalent to **R ≥ 0, Y > 0, X ≥ 0, Q = 0, X = R**

Assignments



Loops

- Unlike other statements, a loop, like the one in the example, has to deal with multiple incoming flows of control
  - That is, there are two ways of entering the loop
    - One from the start of the program
    - One after going around the loop at least once
- The statements inside the loop must be true in both circumstances
  - In fact, they need to be true no matter how many times the loop is executed

Loops - 2

- For this reason, they are called *loop invariants*
  - Normally, you think of loops behaving differently on each iteration
  - But the assertion has to stay the same
- That is, the loop invariant has to generalize over all iterations
  - They can do this because they are expressed in terms of the loop variables
  - The analyst has to invent this generalization

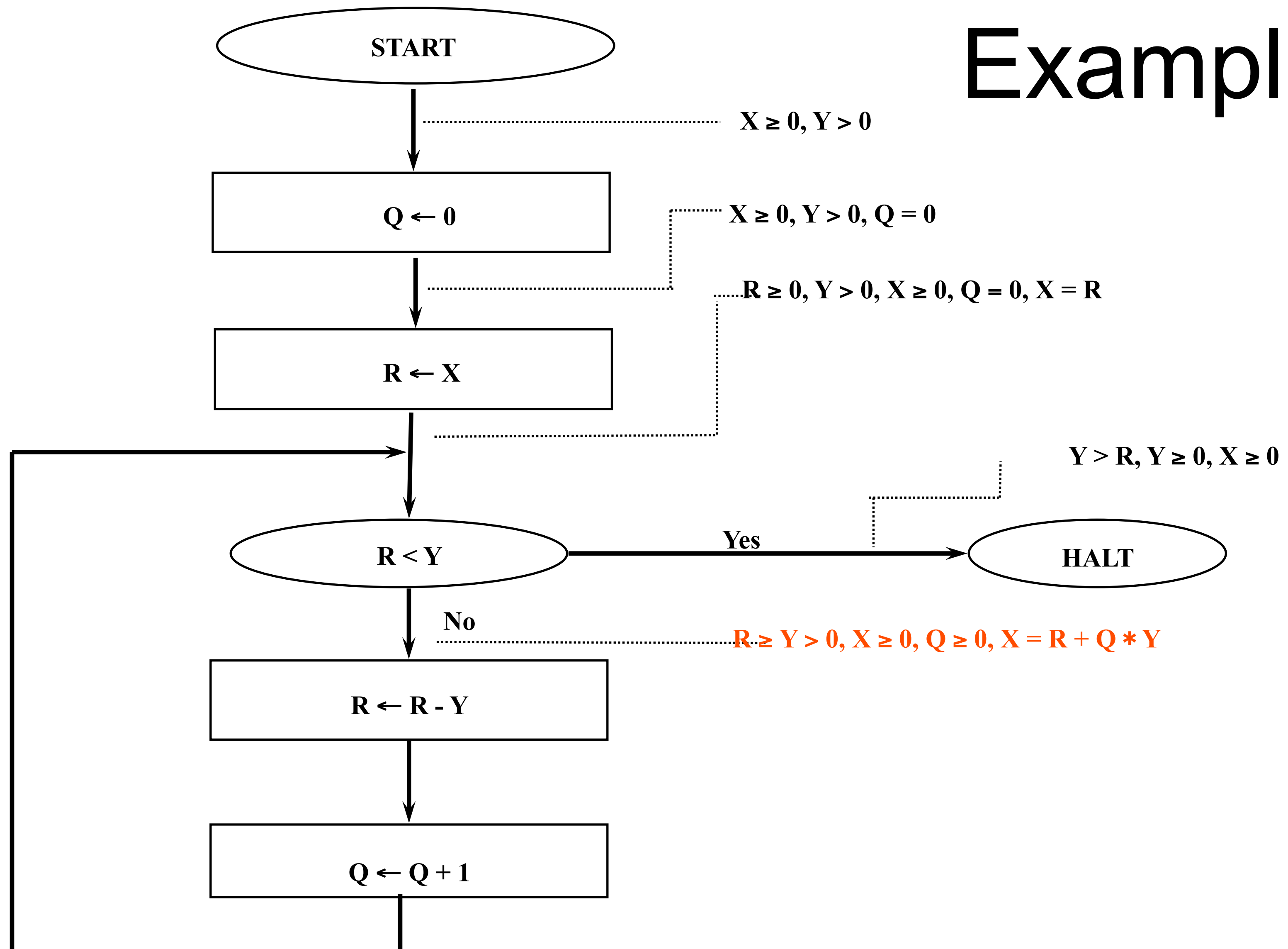
More on Loop Invariants

- A loop invariant has to satisfies three properties
  - It must be true the first time execution reaches it
  - If it is true after some number (n) of iterations, it must be true after n + 1 iterations
  - It must be strong enough to imply the postcondition

Loop Invariants - 2

- Let’s start with the postcondition we have to establish
  - $Y > R \geq 0 \wedge X \geq 0 \wedge Q \geq 0 \wedge X = Q * Y + R$
- We already have $Y > 0$, $Y > R$, $X \geq 0$
- Let's try
  $R \geq Y > 0, X \geq 0, Q \geq 0, X = R + Q * Y$

Example



Invariants - 2

- Let's try our first test. Is the invariant true the first time through?
  - Condition before entering the loop
    $\mathbf{R \geq 0, Y > 0, X \geq 0, Q = 0, X = R}$
  - Loop invariant
    $\mathbf{R \geq Y > 0, X \geq 0, Q \geq 0, X = R + Q * Y}$
  - $\mathbf{R \geq 0}$ and $\mathbf{Y > 0}$ and $\mathbf{R \geq Y}$ implies $\mathbf{R \geq Y > 0;}$ so we are okay so far
  - **X** still non-negative
  - If **Q** equal to **0**, then it is certainly greater than or equal to it
  - Finally, if $\mathbf{Q = 0}$ and $\mathbf{X = R}$ then **X** does equal
    $\mathbf{R + Q * Y = R + 0 * Y = R}$

Assignments Again

- The algorithm includes the assignment statement **R ← R - Y** inside the loop
- What is interesting about this statement is that the variable on the left hand side (**R**) also occurs on the right hand side
- If we naively state that the postcondition is **{R = R - Y},** we would get nonsense
- Instead, we must introduce a little more notation and perform some algebraic manipulations

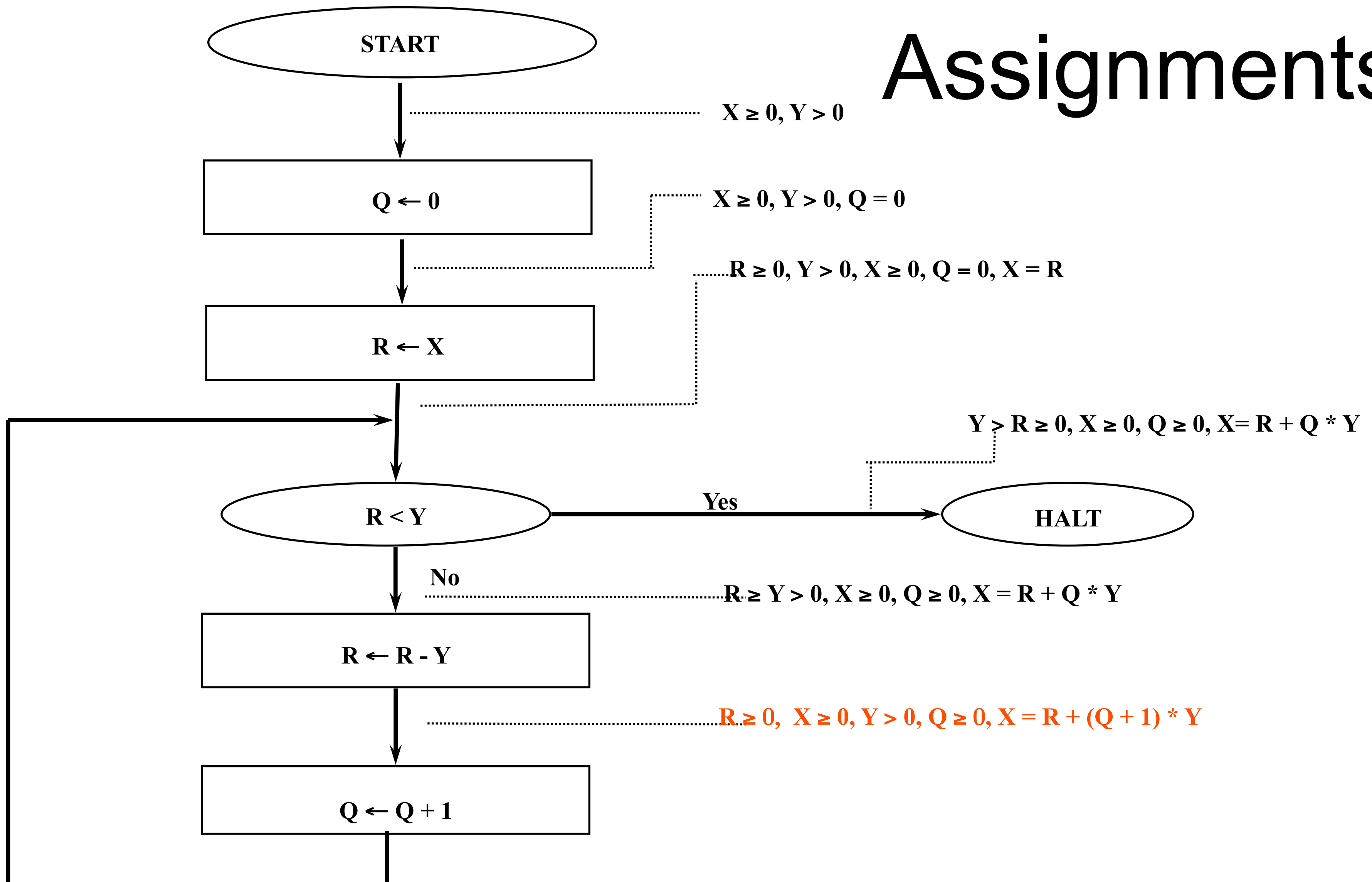
Assignments - 2

- Assume that the precondition for the assignment statement is **{X = R + Q * Y}** and the assignment statement **R ← R - Y**
- First, using the assignment statement, annotate the left hand **R** with a prime **(R')**
  - **R'** can be read as "the value of **R** after the assignment”

Assignments - 3

- Now, solve for **R** in terms of **R':**
  $\mathbf{R' = R - Y \Rightarrow R' + Y = R}$
- Substitute this expression **(R' + Y)** into the precondition for all occurrences of **R**:
  $\mathbf{\{X = (R' + Y) + Q * Y\}}$
- Simplify to produce the postcondition and drop the prime:
  $\mathbf{\{X = R + (Q + 1) * Y\}}$
- The general rule is to solve for the variable without the apostrophe and plug that expression into the precondition

More Assignments



Last Assignment

- Let's try this procedure on the last assignment statement in the algorithm $\mathbf{Q \leftarrow Q + 1}$
- The precondition of the assignment is
  $\mathbf{R \geq 0,\ \ X \geq 0,\ Y > 0,\ Q \geq 0,\ X = R + (Q + 1) * Y}$
- Set $\mathbf{Q' = Q + 1}$
- Solve for **Q:** $\mathbf{Q = Q' - 1}$
- Substitute into precondition:
  $\mathbf{R \geq 0,\ \ X \geq 0,\ Y > 0,\ (Q' - 1) \geq 0,\ X = R + ((Q' - 1) + 1) * Y}$
- Simplify
  $\mathbf{R \geq 0,\ \ X \geq 0,\ Y > 0,\ Q > 0,\ X = R + Q * Y}$

Last Assignment

START

$X \geq 0, Y > 0$

$Q \leftarrow 0$

$X \geq 0, Y > 0, Q = 0$

$R \leftarrow X$

$R \geq 0, Y > 0, X \geq 0, Q = 0, X = R$

$R < Y$

Yes

HALT

$Y > R \geq 0, X \geq 0, Q \geq 0, X = R+Q*Y$

No

$R \geq Y > 0, X \geq 0, Q \geq 0, X = R + Q * Y$

$R \leftarrow R - Y$

$R \geq 0, X \geq 0, Y > 0, Q \geq 0, X = R + (Q + 1) * Y$

$Q \leftarrow Q + 1$

$R \geq 0, X \geq 0, Y > 0, Q > 0, X = R + Q * Y$

Generalization

- We now have two preassertions for the loop

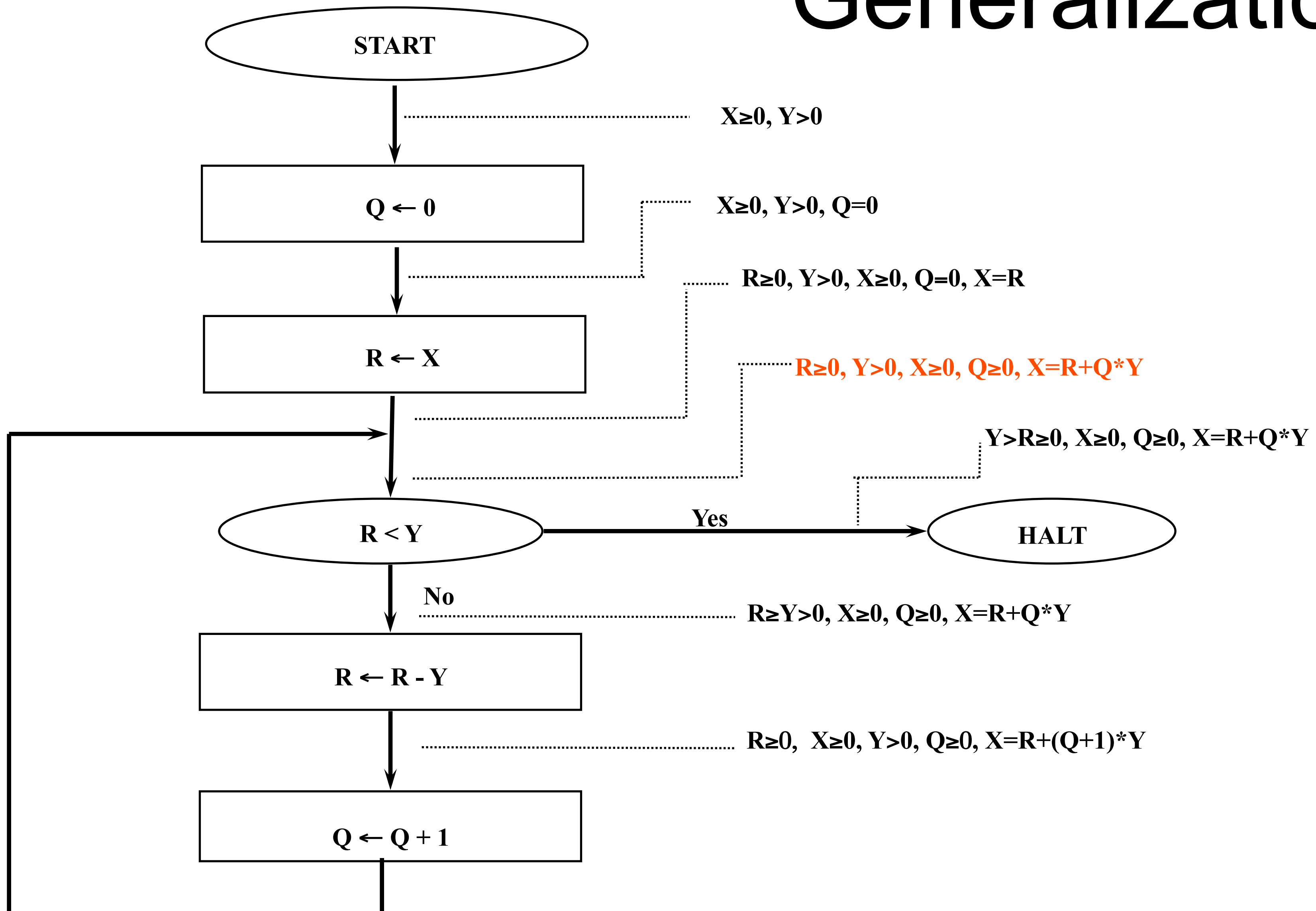
  R >= 0, Y > 0, X >= 0, Q = 0, X = R

  R >= 0, Y > 0, X >= 0, Q > 0, X = R + Q * Y

- We need to find a logical expression that generalizes over both:

  R >= 0, Y > 0, X >= 0, Q >= 0, X = R + Q * Y

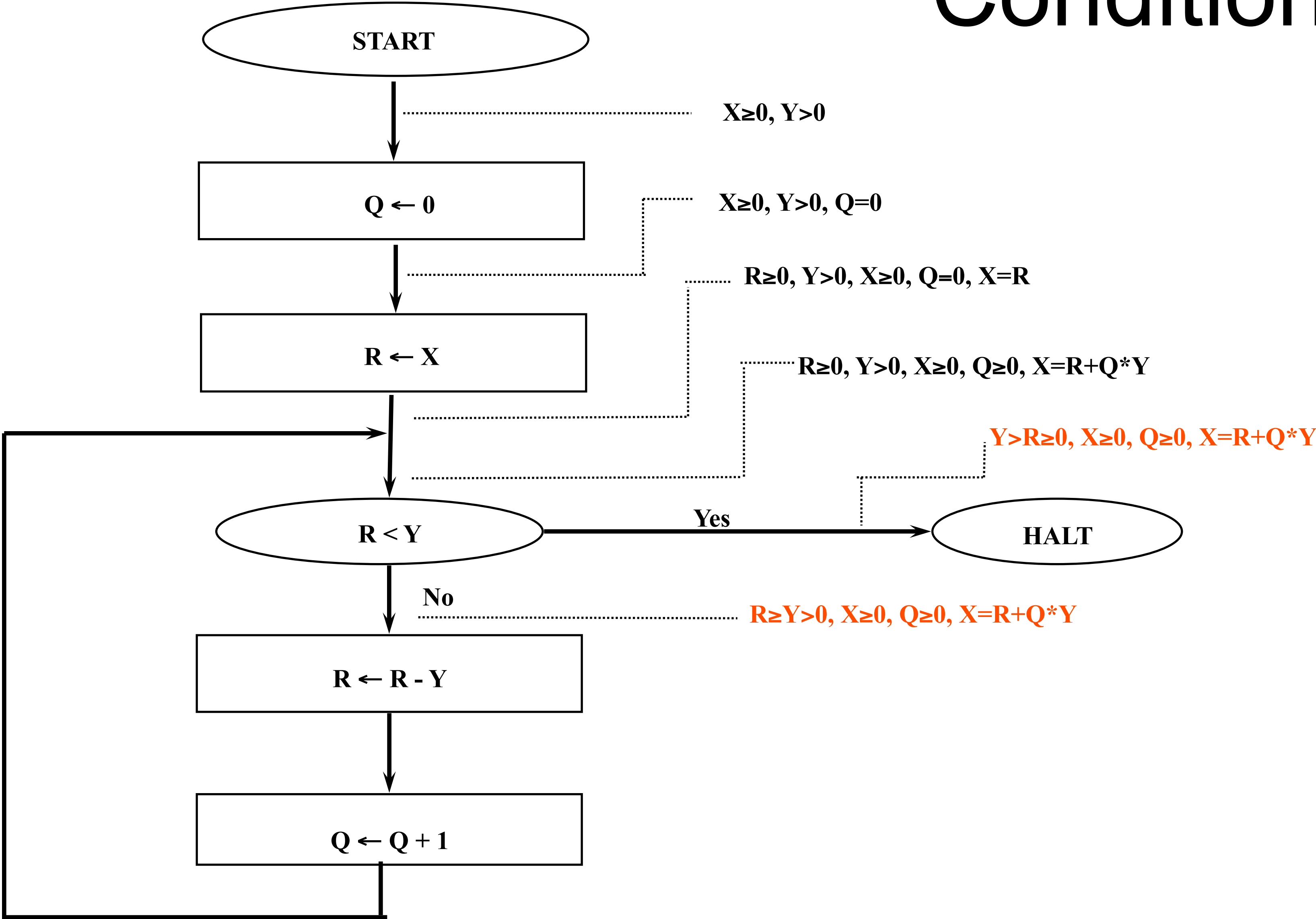
Generalization



Conditionals

- A conditional statement, like an assertion, describes a possible program state
- Execution branches leading out of a conditional statement can be labeled with an assertion corresponding to the truth or falsity of the Boolean condition tested
- In the example, the conditional tests the value of the expression **R < Y**
- On the true (**Yes**) branch leading out from this test, **R < Y,** can be conjoined with the assertions coming into the conditional
  - Similarly for the **No** branch

Conditional



Implications

- Notice that the postcondition of the last assignment labels the arc that returns to the top of the loop
- We can now make our second test on the loop invariant: if the loop has successfully executed n times, will the invariant hold on the $n + 1^{st}$?
- The postcondition on the nth execution (or any execution) is $\mathbf{R \geq 0,\ X \geq 0,\ Y > 0,\ Q > 0,\ X = R + Q * Y}$
- The loop invariant is $\mathbf{R \geq Y > 0,\ X \geq 0,\ Q \geq 0,\ X = R + Q * Y}$
- Surely $\mathbf{Q > 0}$ implies $\mathbf{Q \geq 0}$
- And $\mathbf{R \geq 0,\ Y > 0,}$ and the loop condition $\mathbf{R \geq Y}$ imply $\mathbf{R \geq Y > 0}$
- So the second test of our loop invariant is passed

Third Test

- It is easy to come up with invariants. After all, 1 + 1 = 2 is a loop invariant. It is true for every execution of any loop
  - But it is not much good in proving programs
- We need to have loop invariants that imply the program's postconditions
  - This is just another way of saying that the loop computation has to contribute to the producing the intended program result

Third Test - 2

- For the example program, the result of the loop is
  $\mathbf{R \geq 0, \; X \geq 0, Y > 0, Q > 0, X = R + Q * Y}$
- We also know from the conditional that $\mathbf{R < Y}$
- The program post condition is
  $\mathbf{Y > R \geq 0, X \geq 0, Q \geq 0, X = Q * Y + R}$
- So the third test is passed as well

Summary of Example

- Preconditions for successful execution
- Postconditions
- Examination of all possible paths
- Assignment
- Conditionals
- Loop
  - Invariant
  - First execution
  - Induction
  - Strong enough

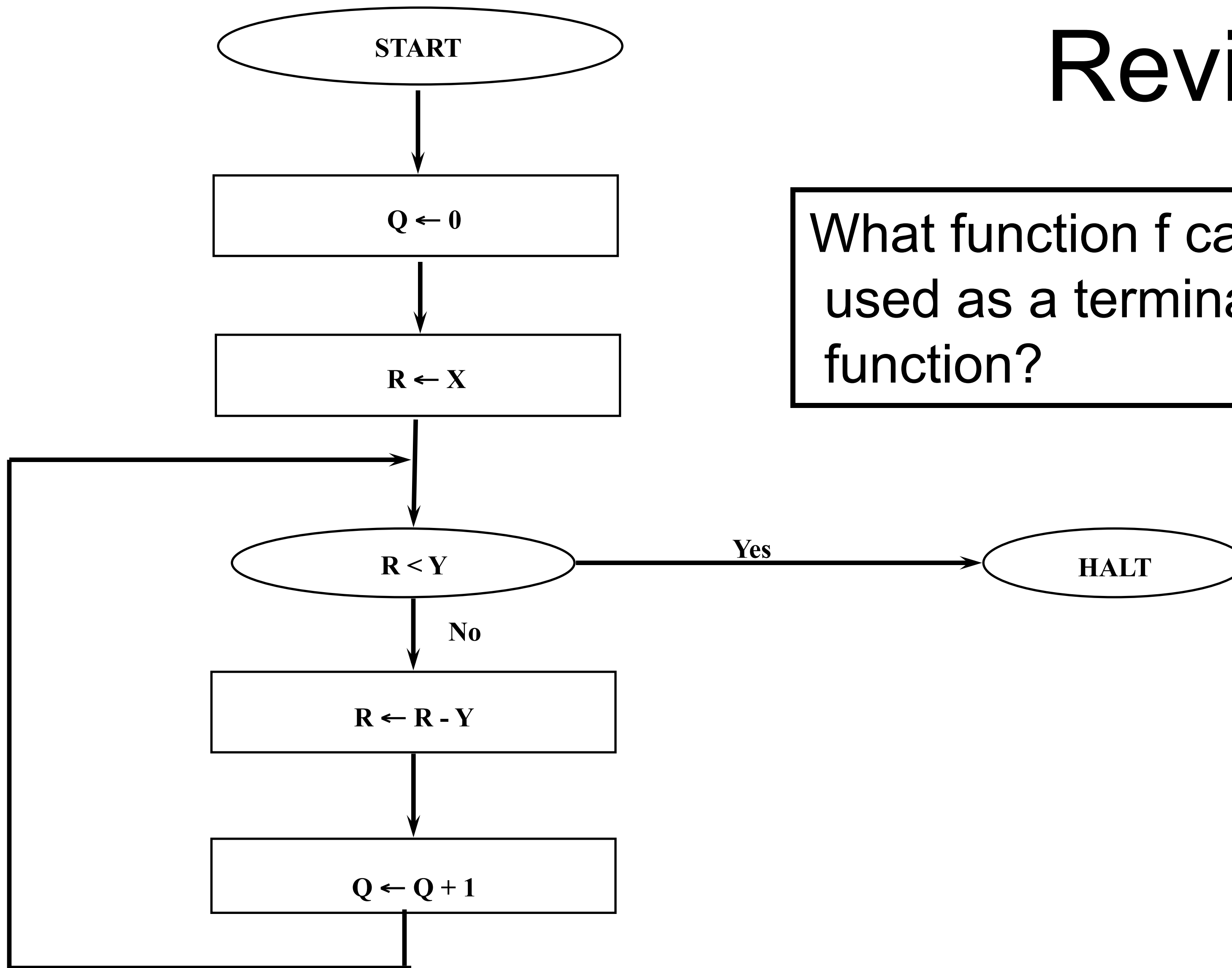
Termination

- There is one other detail to clear up
- We have proved that the program is correct (matches its specification), assuming that it terminates
- We have to also prove termination
- Assignments and conditions always progress, so loops are the concern
- The way to show that loops terminate is to show that they always make progress

Termination of Loops

- You can show that loops terminate as follows
- Define a function with the following properties
  - Its domain is the set of program variables
  - Its range is the integers
  - The value of computed by the function gets smaller on every iteration
  - When the loop exits, the function value is negative
- That is, you can rely on the following property of the integers
  - Any constantly decreasing series of integer values must eventually become negative

Example Revisited



What function f can be used as a termination function?

Termination Example

- Termination function: $f(X, Y, Q, R) = R - Y$
- On the **No** branch:
  - $R \geq Y \Rightarrow R - Y \geq 0 \Rightarrow f$ is always non-negative
  - $R \leftarrow R - Y$ gets executed on every iteration
  - R and Y are both positive at all times
  - Hence, R gets reduced on every loop iteration
  - Consequently f gets reduced on every loop iteration
- On the **Yes** branch: $R < Y \Rightarrow R - Y < 0 \Rightarrow f$ is negative

```
Q ← 0
R ← X
while (R ≥ Y)
   R ← R − Y
   Q ← Q + 1
```

Another Exercise
What Does this Program Do?

Assume B is a sorted Vector of ints of length n

```
I := 1;
P := 1;
while (I != N)
     if (B[I] != B[I - P])
          I := I + 1;
     else
          I := I + 1;
          P := P + 1;
     end if;
end while;
```

Prove it formally