CSC236 Week 8

Larry Zhang

Recap

- We started talking about **program correctness**
- We talked about the correctness of **recursive** programs
  - show the correctness for each program path
  - when having recursive call, assume the recursive calls satisfies the postcondition, then show that the original function call satisfies the postcondition
  - essentially using **induction**

Programs with **loops**

- Loops are harder to analyze
  - For recursive programs, we know the exact sequence of steps of each program path (recursive calls treated as one step), so we can analyze them directly step by step.
  - For programs with loops, we don’t know the exact sequence anymore.
    - We don’t know how many iterations the loop will run
    - We don’t even know whether the loop terminates or not
  - We have to give loops some special treatment
  - Give a separate correctness argument for each loop in the program

Example 1: “Average”

Compute the average of list A

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

How do we argue if this program is correct or not?

Some terminology

```
while E:
    S
```

```
while i < len(A):
    sum += A[i]
    i += 1
```

If we pick the right invariant, it will help proving the correctness of the program with the loop.

- **E** is called the **loop guard** (e.g., `i < len(A)`)
- **S** is called the **loop body** (one or more statements)
- A **loop invariant** gives a relationship between variables
  - it's a **predicate** with the variables being the parameters.
    - e.g., **Inv(i, sum): sum = \sum from A[1] to A[i]**
  - **Requirements** on loop invariant (otherwise it's not an invariant)
    - The invariant must hold prior to the first iteration (i.e., before entering the loop)
    - Assuming that the **invariant** and the **guard** are both true, the invariant must remain true after one **arbitrary** loop iteration

Invariant for summing loop

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

Here is the invariant for this loop:

$$Inv(i, sum) : 0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

The invariant means three things

- ***i*** is at least 0
- ***i*** is at most ***len(A)***
- ***sum*** is the sum of the first ***i*** elements of ***A***

Prove the correctness of a program with loop

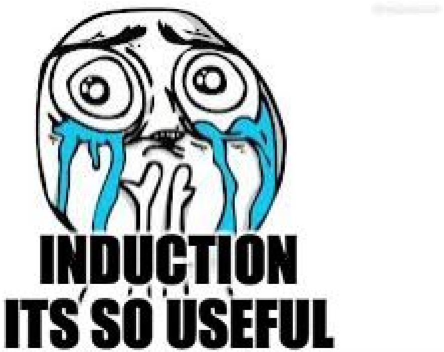
It's another application of induction

- **Base case**: Argue that the loop invariant is true when the loop is reached
- **Induction Step**:
    - assume that the invariant and guard are true at the end of an arbitrary iteration (**induction hypothesis P(n-1)**)
    - show that the invariant remains true after one iteration (**P(n)**)
- **Check postcondition**: Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's postcondition.

Negation of the loop guard because we want to check what happens after we **exit** the loop.

Quick Interlude: **Induction**'s applications so far

- Prove the runtime of recursive programs
- Prove the correctness of recursive programs
- Prove the correctness of loop invariants



Now let’s write the proof for the correctness of AVG

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

$$Inv(i, sum) : 0 \leq i \leq len(A) \land sum = \sum_{k=0}^{i-1} A[k]$$

1. Base case

$$Inv(i, sum) : 0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

Argue that the loop invariant is true when the loops is reached

- When the loop is reached, $i = 0$ and $sum = 0$
- $Inv(0,0) : 0 \leq 0 \leq len(A) \wedge 0 = \sum_{k=0}^{-1} A[k]$
- Both of these conjuncts are true

Base case done

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

2. Induction Step

$$Inv(i, sum): 0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

Assume that the invariant and guard are true at the end of an arbitrary iteration, show that the invariant remains true after one iteration.

- At the end of an arbitrary iteration ***i***, the invariant being true means that ***sum*** is the sum of first ***i*** elements of **A**.
- After one more iteration, we add ***A[i]*** to ***sum,*** therefore ***sum*** becomes the sum of the first ***i+1*** elements of **A.**
- To "fix this", we must increase ***i*** by 1
- that's exactly what the loop does
- so loop invariant **remains true**

This argument is **informal**, we need to make it more **formal.**

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

2. Induction Step (**formal**)

$$Inv(i, sum) : 0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

Assume that the invariant and guard are true at the end of an arbitrary iteration, show that the invariant remains true after one iteration.

We use **subscript 0** for values **before** iteration, and **1** for values **after** iteration.

We assume two things, the **invariant** and the **loop guard**

- $Inv(i_0, sum_0) : \quad 0 \leq i_0 \leq len(A) \wedge sum_0 = \sum_{k=0}^{i_0-1} A[k]$
- $i_0 < len(A)$

We must prove

$$Inv(i_1, sum_1) : \quad 0 \leq i_1 \leq len(A) \wedge sum_1 = \sum_{k=0}^{i_1-1} A[k]$$

We will prove the two conjuncts separately

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

2. Induction Step (**formal**)

- $Inv(i_0, sum_0): \quad 0 \leq i_0 \leq len(A) \wedge sum_0 = \sum_{k=0}^{i_0-1} A[k]$
- $i_0 < len(A)$

$Inv(i_1, sum_1): \quad 0 \leq i_1 \leq len(A) \wedge sum_1 = \sum_{k=0}^{i_1-1} A[k]$

Conjunct 1: $0 \leq i_1 \leq len(A)$

- From the loop body, we have the relationship $i_1 = i_0 + 1$
- Substituting, we want to prove that $0 \leq (i_0 + 1) \leq len(A)$
- The first part is true because $0 \leq i_0$ from the invariant
- The second is true because $i_0 < len(A)$ from the loop guard

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

2. Induction Step (**formal**)

- $Inv(i_0, sum_0): \quad 0 \le i_0 \le len(A) \wedge sum_0 = \sum_{k=0}^{i_0-1} A[k]$
- $i_0 < len(A)$

$$Inv(i_1, sum_1): \quad 0 \le i_1 \le len(A) \wedge sum_1 = \sum_{k=0}^{i_1-1} A[k]$$

Conjunct 2: $sum_1 = \sum_{k=0}^{i_1-1} A[k]$

The loop body gives us $sum_1 = sum_0 + A[i_0]$ and $i_1 = i_0 + 1$.

$$sum_1 = sum_0 + A[i_0] \qquad \text{(from loop body)}$$

$$= \left(\sum_{k=0}^{i_0-1} A[k]\right) + A[i_0] \qquad \text{(by } Inv(i_0, sum_0)\text{)}$$

$$= \sum_{k=0}^{i_0} A[k] \quad = \sum_{k=0}^{i_1-1} A[k]$$

Conjunct 1 & 2 proven, induction step done.

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

Recap of steps: Prove the correctness of a program with loop

It's another application of induction

- **Base case**: Argue that the loop invariant is true when the loop is reached
- **Induction Step**:
  - assume that the invariant and guard are true at the end of an arbitrary iteration (**induction hypothesis P(n-1)**)
  - show that the invariant remains true after one iteration (**P(n)**)
- **Check postcondition**: Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's postcondition.

NEXT STEP

3. Check the postcondition

Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's **postcondition.**

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

The **invariant** and **negation of the guard** together:

$$0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k] \wedge i \geq len(A)$$

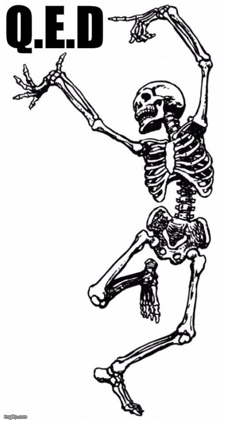
- The only way to satisfy the first and third conjuncts is to conclude that $i = len(A)$

$$i = len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

- So sum is the sum of all elements of A

The code then divide sum by len(A), which gives us the average of the list.

Postcondition satisfied!



Recap of steps: Prove the correctness of a program with loop

It's another application of induction

- **Base case**: Argue that the loop invariant is true when the loop is reached
- **Induction Step**:
  - assume that the invariant and guard are true at the end of an arbitrary iteration (**induction hypothesis P(n-1)**)
  - show that the invariant remains true after one iteration (**P(n)**)
- **Check postcondition**: Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's postcondition.

Example 2: “Multiplication”

A multiplication algorithm

```
def mult(a,b):
    ’’’
    Pre: a and b are natural numbers
    Post: returns a * b
    ’’’
    x = a
    y = b
    total = 0
    while x > 0:
       if x % 2 == 1:
          total = total + y
       x = x // 2
       y = y * 2
    return total
```

- It’s not obvious why this algorithm would work correctly.
- But not to worry, because we know how to prove it!

Let do some tracing first

```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
        if x % 2 == 1:
            total = total + y
        x = x // 2
        y = y * 2
    return total
```

x	y	total
18	3	0
9	6	0
4	12	6
2	24	6
1	48	6
0	96	54
end		

What is **NOT changing** across different lines?

The invariant

$$Inv(x, y, total) : 0 \leq x \wedge total + x * y = a * b$$

Coming up with this invariant requires some good understanding of the loop code:

- Each step we divide ***x*** by ***2***, and multiple ***y*** by ***2***, so ***x*y*** should be roughly the same as ***a*b***
- But when ***x*** is odd, ***x // 2*** lose a ***1***, in terms of ***x*y*** we lose a ***y***.
- That ***y*** is added into ***total*** by the code

```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
        if x % 2 == 1:
            total = total + y
        x = x // 2
        y = y * 2
    return total
```

Recap of steps: Prove the correctness of a program with loop

It's another application of induction

- **Base case**: Argue that the loop invariant is true when the loop is reached
- **Induction Step**:
  - assume that the invariant and guard are true at the end of an arbitrary iteration (**induction hypothesis P(n-1)**)
  - show that the invariant remains true after one iteration (**P(n)**)
- **Check postcondition**: Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's postcondition.

$$Inv(x, y, total) : 0 \leq x \wedge total + x * y = a * b$$

1. Base case

Argue that the loop invariant is true when the loops is reached

- When the loop is reached, $x = a$, $y = b$, and $total = 0$
- $Inv(a, b, 0) : 0 \leq a \wedge 0 + x * y = x * y$
- The first conjunct is true from the precondition
- The second conjunct is true by definition

Base case done

```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
       if x % 2 == 1:
          total = total + y
       x = x // 2
       y = y * 2
    return total
```

$$Inv(x, y, total) : 0 \le x \wedge total + x * y = a * b$$

2. Induction Step

Assume that the invariant and guard are true at the end of an arbitrary iteration, show that the invariant remains true after one iteration.

We use **subscript 0** for values **before** iteration, and **1** for values **after** iteration.

We assume two things, the **invariant** and the **loop guard**

Try it yourself!

```
def mult(a,b):
  '''
  Pre: a and b are natural numbers
  Post: returns a * b
  '''
  x = a
  y = b
  total = 0
  while x > 0:
     if x % 2 == 1:
        total = total + y
     x = x // 2
     y = y * 2
  return total
```

$Inv(x, y, total) : 0 \leq x \wedge total + x * y = a * b$

2. Induction Step

$$0 \leq x_0 \wedge total_0 + x_0 * y_0 = a * b \wedge x_0 > 0$$

$$x_1 = \lfloor x_0/2 \rfloor \qquad y_1 = y_0 * 2$$

first conjunct: $x_1 \geq 0$ because $x_0 \geq 0$

second conjuct, need to show:

$$total_1 + x_1 * y_1 = a * b$$

need 2 cases: x_0 is even or odd

```
def mult(a,b):
  '''
  Pre: a and b are natural numbers
  Post: returns a * b
  '''
  x = a
  y = b
  total = 0
  while x > 0:
     if x % 2 == 1:
        total = total + y
     x = x // 2
     y = y * 2
  return total
```

$Inv(x, y, total) : 0 \leq x \land total + x * y = a * b$

2. Induction Step

Case 1: x_0 is even

$$x_1 = x_0/2$$

$$y_1 = y_0 * 2$$

$$total_1 = total_0$$

$$\begin{aligned} & total_1 + x_1 * y_1 \\ = \; & total_0 + (x_0/2) * (y_0 * 2) \\ = \; & total_0 + x_0 * y_0 \; = a * b \end{aligned}$$

```python
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
        if x % 2 == 1:
            total = total + y
        x = x // 2
        y = y * 2
    return total
```

$Inv(x, y, total) : 0 \le x \wedge total + x * y = a * b$

2. Induction Step

Case 2: x_0 is odd

$x_1 = (x_0 - 1)/2 \qquad y_1 = y_0 * 2$

$total_1 = total_0 + y_0$

$$
\begin{aligned}
& total_1 + x_1 * y_1 \\
= & (total_0 + y_0) + (x_0 - 1)/2 * y_0 * 2 \\
= & (total_0 + y_0) + (x_0 - 1) * y_0 \\
= & total_0 + y_0 + x_0 * y_0 - y_0 \\
= & total_0 + x_0 * y_0 = a * b
\end{aligned}
$$

```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
        if x % 2 == 1:
            total = total + y
        x = x // 2
        y = y * 2
    return total
```

$Inv(x, y, total) : 0 \le x \wedge total + x * y = a * b$

3. Check the postcondition

Argue that the **invariant** and the **negation of the loop guard** together let us conclude the program's **postcondition.**

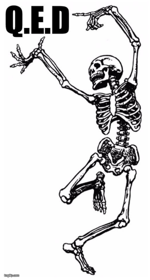
The **invariant** and **negation of the guard** together:

$$Inv(x, y, total) : 0 \le x \wedge total + x * y = a * b \ \wedge x \le 0$$

from conjunct 1 and 3: $x = 0$

then conjunct 2 becomes: $total = a * b$

Postcondition satisfied!




```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
        if x % 2 == 1:
            total = total + y
        x = x // 2
        y = y * 2
    return total
```

We have been ignoring an important issue ...

Loop Termination

Loop Termination

- A correct program must terminate
- so our correctness proof must prove that
- but the loop-invariant-based proof that we have been using does NOT tell us anything about loop termination

Loop invariants does NOT tell about termination

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        pass
    return sum / len(A)
```

- Any invariant that is true when the loop is reached will stay true
- But the loop never terminates
- And the program is not correct

So, how do we prove loop termination

Use **Loop Variants** (not invariants)

- Identify a loop variant **v** that has the following two properties
  a. **v** is decreased by each iteration of the loop
  b. The invariant and the guard together imply that **v >= 0**
- If **v** decreases on each iteration yet cannot drop below 0, then we can conclude that at some point the loop must terminate

Termination of Summing Loop

```
def avg(A):
    '''
    Pre: A is a non-empty list of numbers
    Post: Returns the average of the numbers in A
    '''
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)
```

$$Inv(i, sum) : 0 \leq i \leq len(A) \wedge sum = \sum_{k=0}^{i-1} A[k]$$

What should the variant be?

- Should it be ***i*** ?
    - No, because ***i*** increases on each iteration
- ***len(A) - i***
    - It decreases on each iteration, because ***i*** increases
    - From the invariant we know that ***i <= len(A)***, i.e., ***len(A) - i >= 0***
    - Good choice of variant

Termination of Multiplication Algorithm

```
def mult(a,b):
    '''
    Pre: a and b are natural numbers
    Post: returns a * b
    '''
    x = a
    y = b
    total = 0
    while x > 0:
       if x % 2 == 1:
          total = total + y
       x = x // 2
       y = y * 2
    return total
```

The variant can be **x**:

- x decreases on each iteration
- by the loop guard implies that x > 0

So `mult` terminates.

Another Example

```
def term_ex(x,y):
    '''
    Pre: x and y are integers >= 0
    '''
    a = x
    b = y
    while a > 0 or b > 0:
        if a > 0:
            a -= 1
        else:
            b -= 1
    return x * y
```

Home Exercise: Prove the invariant

Assume this loop invariant: $a \geq 0 \wedge b \geq 0$

What's the variant?

- x ?
  - no good, it never changes.
- a ?
  - no good, it does NOT decrease on **each** iteration
    - when a=0 and b>0
- b ?
  - no good, it does NOT decrease on **each** iteration
    - when a > 0
- a+b ?
  - **GOOD**
  - always decreases, because one of a and b must decrease on each iteration
  - a+b >= 0 according to the invariant

Yet Another Example

```
def collatz(n):
    ’’’
    Pre: n is a natural number
    ’’’
    curr = n
    while curr > 1:
        if curr is even:
            curr = curr // 2
        else:
            curr = 3 * curr + 1
```

What’s the variant to prove this program’s termination?

Actually, nobody knows yet. It is an open question in mathematics whether it terminates on all inputs or not.

Summary

- We learned a formal way to prove the correctness of programs with loops
  - **Loop Invariant**
- We learned a formal way to prove loop termination
  - **Loop variant**
- Together with what we learned last week, now you have the ability to **formally analyze** the correctness of pretty much any program.
- With this analytical ability, you will be more likely to write correct programs, like the pros do.

Next week

- New topic: Language and Regular Expressions