Iterative Correctness

2025-07-09

Recap

- Correctness: Precondition $\Longrightarrow$ Postcondition
- For recursive algorithms, proof of correctness is done by induction on the input size.
- Karatsuba's Algorithm for multiplication.
  - $n^{\log_2(3)} \leq n^{1.59}$ time!

Iterative Algorithms

```
1  i = 0
2  while i < N:
3      # some more code here...
4      i += 1
```

- Convention: After the kth iteration means just before the loop condition is evaluated for the $k+1$ time
- After the kth iteration is the same as before the $k+1$th iteration

Example

More multiplication!

- Input: (x, y)
- Precondition: $x, y \in \mathbb{N}$
- Postcondition: return xy.

```
1  def mult(x, y):
2      i = 0
3      total = 0
4      while i < x:
5          total = total + y
6          i = i + 1
```

return total.

$y + y + y + y + \ldots + y$

x times.

By induction. This time on the # of iterations.

Define $P(n)$: after the n^{th} iteration

a.) $total_n = ny$

b.) $i_n = n$

WTS $\forall n. P(n)$.

Base case: $total_0 = 0$, $= 0 \cdot y$ ✓

$i_0 = 0$

Inductive step: let $k \in \mathbb{N}$, WTS.

$P(k) \Rightarrow P(k+1)$

if the $k+1$th iteration didn't run, then $P(k+1)$ is vacuously true,

So, suppose the $k+1$th iteration ran.

a) WTS: $total_{k+1} = (k+1) \cdot y$.

$total_{k+1} = total_k + y$

$= ky + y$ (IH). , $P(k).a$.

$= (k+1)y$.

b) WTS $i_{k+1} = k+1$.

$i_{k+1} = i_k + 1$

$= k+1$ (IH). , $P(k).b$.

This completes the induction.

The while condition is: $i < x$

The value of $i_k = k$, so, the condition is true for all $k < x$.

then, after iteration x, $i_x = x$. ($P(x).b$).

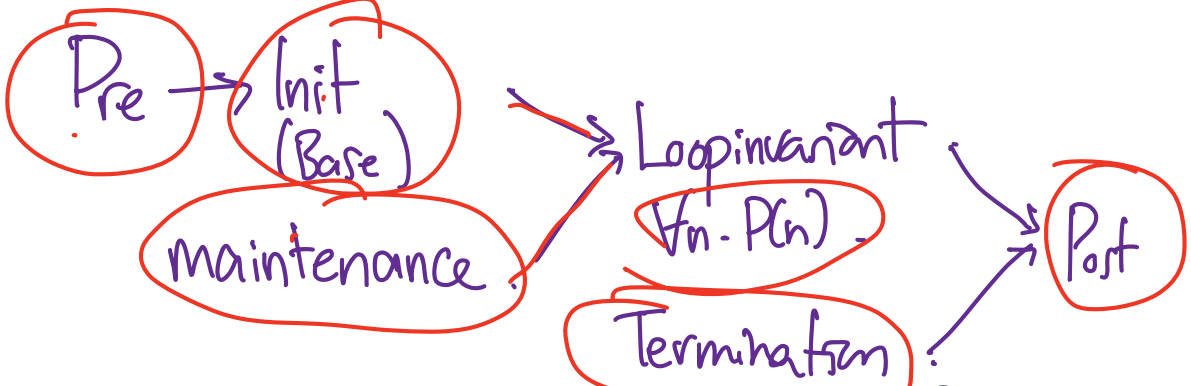
thus $i < x$ is not true

so we the exit the loop after the x^{th} iteration and return $total_x = xy$ by $P(x).a$.

Subscripts

- If x is some variable in the function, use x_i to denote the value of a variable after iteration i
-

General Strategy



Define a loop invariant - some property that is true at the end of every iteration. Call the property $P(i)$. I.e., $P(i)$ holds if the property is true after iteration i. WTS $\forall i \in \mathbb{N} . P(i)$

Prove the following:

- **Initialization**. Show that the loop invariant is true at the start of the loop if the precondition holds.
- **Maintenance**. Show that if the loop invariant is true at the start of any iteration, it is also true at the start of the next iteration.
- **Termination**. Show that the loop terminates and that when the loop terminates, the loop invariant applied to the last iteration implies the postcondition.

Runtime

last time: multiplying n-digit numbers faster than $\Theta(n^{1.59})$.

99.

```
1  def mult(x, y):
2      i = 0
3      total = 0
4      while i < x:
5          total = total + y
6          i = i + 1
```

$99\ldots9$ (n)

$10^n - 1$

$\approx \Theta(10^n)$.

~~$\Omega(n)$~~ $\Omega(n10^n)$ ☹

fact: the fastest alg for multiplication is $O(n\log(n))$ discovered in 2019.

For Loops ⟺ While Loops

```
1  for i in range(N):
2      # some code here...```
```

is equivalent to

```
1  i = 0
2  while i < N:
3      # some code here...
4      i += 1
```

Proving Termination

P(n)

- Usually, a consequence of the loop invariant.
- The loop invariant implies the loop condition is true after iterations $0, 1, ..., N - 1$
- The loop invariant implies the loop condition is false after iteration N, so the loop exits after the Nth iteration.

Mystery Algorithm

Precondition: $x, y \in \mathbb{N}, y > 0$.

```
1  def mystery(x, y):
2      val = 0
3      c = 0
4      while val < x:
5          val = val + y
6          c = c + 1
7      return c
```

$n-1 < x/y \leq n$

$(n-1)y < x \leq ny$

$P(n)$: After iteration n ...

a) $c_n = n$

b) $val_n = ny$

Base case

inductive step:

Initialization: $P(0)$. $c_0 = 0$ $val_0 = 0 = 0 \cdot y$.

Maintenance: $P(k) \Rightarrow P(k+1)$.

(IH).

a.) $c_{k+1} = c_k + 1 = k+1$

b.) $val_{k+1} = val_k + y = ky + y = (k+1)y$.

Termination: $val_n < x$ whenever.

$ny < x$ we exit the loop

when $ny \geq x$. we return $c_n = n$.

P(n): After iteration n ...

a) $c_n = n$

b) $val_n = ny$

while val < x:

⋮

1.) Show alg terminates.

↳ By contradiction, suppose the loop continues indefinitely. In particular, this means that there is an iteration $1000x$. Then $P(1000x).b$ implies $val_{1000x} = 1000xy$. But then, $1000xy \geq x$ so this was the last iteration → contration.

2.) ⇒ the alg terminates after iteration N for some N.

⇒ all iterations prior to N, satisfied the loop condition, and # after the N^{th} iteration, the loop condition was false.

$val_0 < x$, $val_1 < x$, ..., $val_{N-1} < x$, $val_N \geq x$.

$0 < x$, $y < x$, ..., $(N-1)y < x$, $Ny \geq x$

$(N-1)y < x \leq Ny$

$\Rightarrow N-1 < \frac{x}{y} \leq N$

$\Rightarrow N = \lceil \frac{x}{y} \rceil$.

Convention

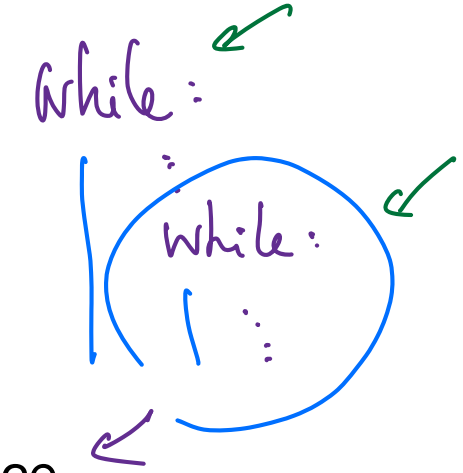
If the predicate $P(n)$ has multiple parts like

a. ...

b. ...

Use $P(n)$.a, $P(n)$.b,... to refer to specific parts of the predicate.

Variations



- One loop after another
  - Prove the correctness of each loop in sequence
- Nested loops
  - “Inside out”. Decompose (or imagine) the inner loop as a separate function. Prove the correctness of that function as a lemma and then prove the correctness of the outer loop. We’ll see some examples in the tutorial.

Another way to prove termination

Descending Sequence

- Define a descending sequence of natural numbers indexed by the iteration number. I.e. $a_1 > a_2 > a_3 > \dots$
- Then, $A = \{a_1, a_2, \dots, \}$ must be a finite set; otherwise, it would be a set of natural numbers with no minimal element, contradicting the Well-Ordering Principle.

Example.

$y > 0$.

```
1  def mystery(x, y):
2      val = 0
3      c = 0
4      while val < x:
5          val = val + y
6          c = c + 1
7      return c
```

$a_n = x + y - val_n$.

Claim: a_n is descending, and a_n is a natural number.

Proof: by induction. $P(n)$: if the n^{th} iteration ran, then $a_n < a_{n-1}$

$\forall n \geq 1$ $P(n)$.

Base case. $a_0 = x+y$.

$a_1 = x+y-y = x$. $a_1 < a_0$.

Inductive $a_k = x+y-val_k = x+y-(val_{k-1}+y)$

$= a_{k-1}+y$.

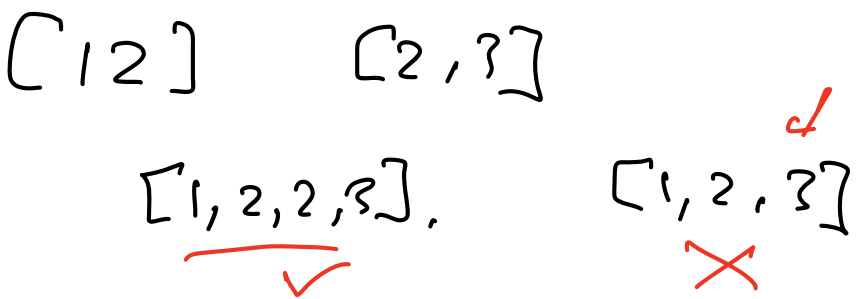
Proofs of Termination

- Most of the time, the LI will imply termination, saving you from having to do another induction proof. I prefer this method.
- However, it is easier to define a descending sequence of natural numbers in some cases - we'll see some examples in the tutorial.

Merge

```
1  def merge(x, y):
2      l = []
3      while len(x) > 0 or len(y) > 0:
4          if len(x) > 0 and len(y) > 0:
5              if y[0] <= x[0]:
6                  l.append(y.pop(0)) # 1.
7              else:
8                  l.append(x.pop(0)) # 2.
9          elif len(x) == 0:
10             l.append(y.pop(0)) # 3.
11         else:
12             l.append(x.pop(0)) # 4.
13     return l
```

- Inputs: x, y are lists of sortable elements.
- Precondition: x, y are sorted lists
- Postcondition: returns a sorted list consisting of all the elements in x and y



Counters

↪ collections.Counter.

- If $l \in \text{List}[X]$, then $\text{Counter}(l)$ is a mapping of elements of l to the number of times they appear.
- E.g. $\text{Counter}([1,2,3,2,1,1,1,4]) = \{1 : 4,\ 2 : 2,\ 3 : 1,\ 4 : 1\}$
- You can think of $\text{Counter}(l)$ as a function from $X \to \mathbb{N}$

Counter(l)(1) = 4.

Merge

```
1  def merge(x, y):
2      l = []
3      while len(x) > 0 or len(y) > 0:
4          if len(x) > 0 and len(y) > 0:
5              if y[0] <= x[0]:
6                  l.append(y.pop(0)) # 1.
7              else:
8                  l.append(x.pop(0)) # 2.
9          elif len(x) == 0:
10             l.append(y.pop(0)) # 3.
11         else:
12             l.append(x.pop(0)) # 4.
13     return l
```

- Inputs: x, y are lists of sortable elements.
- Precondition: x, y are sorted lists
- Postcondition:

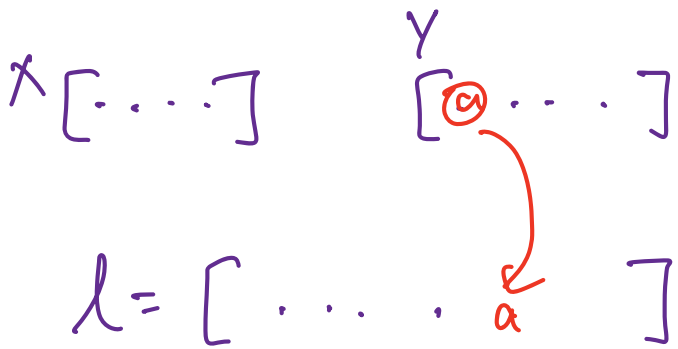
return l, s.t.

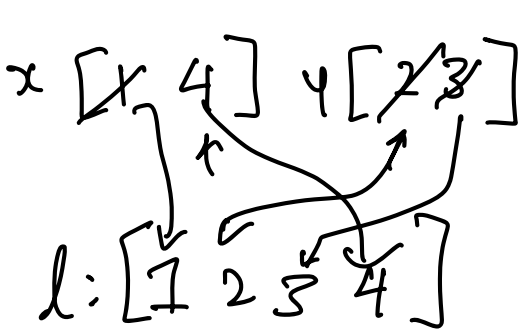
① l sorted.

② Counter(l) = Counter($x + y$)

concatenation.

```python
def merge(x, y):
    l = []
    while len(x) > 0 or len(y) > 0:
        if len(x) > 0 and len(y) > 0:
            if y[0] <= x[0]:
                l.append(y.pop(0)) # 1.
            else:
                l.append(x.pop(0)) # 2.
        elif len(x) == 0:
            l.append(y.pop(0)) # 3.
        else:
            l.append(x.pop(0)) # 4.
    return l
```





Counter(l) = Counter(x0+y0).

P(n): after iteration n:

a.) len(l_n) = n

b.) l_n is sorted.

c) len(x_n) + len(y_n) + len(l_n) = len(x_0) + len(y_0).

d) Counter(x_n + y_n + l_n) = Counter(x_0 + y_0)

L.

Takeaways

- It's normal for the loop invariant to have many parts!
- If you're trying to prove a loop invariant and you get stuck and wish some other property holds, try adding what you need as part of the loop invariant.
- For example, it's common for part 4 of a loop invariant to imply part 1 of the loop invariant.

Summary: Correctness

- If the algorithm is recursive, prove correctness directly by induction on the size of the input.
- For algorithms with loops, prove the correctness of the loop by defining a Loop Invariant, proving the Loop Invariant, and showing that the Loop Invariant holds at the end of the algorithm implies the postcondition.