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
```

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

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

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

- 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
```

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.

```
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
```

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

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

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:

```
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
```

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.